



*The Apple IIc  
Reference Manual  
Volume 2*

*The Apple IIc*



---

## **Customer Satisfaction**

If you discover physical defects in the manuals distributed with an Apple product or in the media on which a software product is distributed, Apple will replace the documentation or media at no charge to you during the 90-day period after you purchased the product.

In addition, if Apple releases a corrective update to a software product during the 90-day period after you purchased the software, Apple will replace the applicable disks and documentation with the revised version at no charge to you during the six months after the date of purchase.

In some countries the replacement period may be different; check with your authorized Apple dealer. Return any item to be replaced with proof of purchase to Apple or an authorized Apple dealer.

---

## **Limitation on Warranties and Liability**

Even though Apple has tested the software described in the manual and reviewed its contents, neither Apple nor its software suppliers make any warranty or representation, either express or implied, with respect to this manual or to the software described in this manual, their quality, performance, merchantability, or fitness for any particular purpose. As a result, this software and manual are sold "as is," and you the purchaser are assuming the entire risk as to their quality and performance. In no event will Apple or its software suppliers be liable for direct, indirect, incidental, or consequential damages resulting from any defect in the software or manual, even if they have been advised of the possibility of such damages. In particular, they shall have no liability for any programs or data stored in or used with Apple products, including the costs of recovering or reproducing these programs or data. Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you.

---

## **Copyright**

This manual and the software (computer programs) described in it are copyrighted by Apple or by Apple's software suppliers, with all rights reserved. Under the copyright laws, this manual or the programs may not be copied, in whole or part, without the written consent of Apple, except in the normal use of the software or to make a backup copy. This exception does not allow copies to be made for others, whether or not sold, but all of the material purchased (with all backup copies) may be sold, given, or lent to another person. Under the law, copying includes translating into another language.

You may use the software on any computer owned by you, but extra copies cannot be made for this purpose. For some products, a multi-use license may be purchased to allow the software to be used on more than one computer owned by the purchaser, including a shared-disk system. (Contact your authorized Apple dealer for information on multi-use licenses.)

---

## **Product Revisions**

Apple cannot guarantee that you will receive notice of a revision to the software described in the manual, even if you have returned a registration card received with the product. You should periodically check with your authorized Apple dealer.

© Apple Computer, Inc. 1984  
20525 Mariani Avenue  
Cupertino, California 95014

Apple, the Apple logo, and ProDOS are trademarks of Apple Computer, Inc.  
Simultaneously published in the United States and Canada. All rights reserved.

---

## **Warning**

This equipment has been certified to comply with the limits for a Class B computing device, pursuant to Subpart J of Part 15 of FCC Rules. Only peripherals (computer input/output devices, terminals, printers, etc.) certified to comply with the Class B limits may be attached to this computer. Operation with non-certified peripherals is likely to result in interference to radio and TV reception.



# *The Apple IIc*

## *Apple IIc Reference Manual* Volume 2



# *Table of Contents*



---

**List of Figures and Tables****ix**

---

**Preface****xiii**

---

**Appendix A****The 65C02 Microprocessor****1**

- 2 A.1 Differences Between 6502 and 65C02
- 2 A.1.1 Differing Cycle Times
- 3 A.1.2 Differing Instruction Results
- 4 A.2 Data Sheet

---

**Appendix B****Memory Map****15**

- 15 B.1 Page Zero
- 19 B.2 Page Three
- 20 B.3 Screen Holes
- 23 B.4 The Hardware Page

---

**Appendix C****Important Firmware Locations****31**

- 31 C.1 The Tables
- 32 C.2 Port Addresses
- 34 C.3 Other Video and I/O Firmware Addresses
- 34 C.4 Applesoft BASIC Interpreter Addresses
- 34 C.5 Monitor Addresses

<b>Appendix D</b>	<b>Operating Systems and Languages</b>	<b>37</b>
37	D.1 Operating Systems	
37	D.1.1 ProDOS	
37	D.1.2 DOS	
38	D.1.3 Pascal Operating System	
38	D.1.4 CP/M	
38	D.2 Languages	
38	D.2.1 Applesoft BASIC	
39	D.2.2 Integer BASIC	
39	D.2.3 Pascal Language	
39	D.2.4 FORTRAN	
<b>Appendix E</b>	<b>Interrupts</b>	<b>41</b>
41	E.1 Introduction	
41	E.1.1 What Is an Interrupt?	
42	E.1.2 Interrupts on Apple II Computers	
43	E.1.3 Interrupt Handling on the 65C02	
43	E.1.4 The Interrupt Vector at \$FFFE	
44	E.2 The Built-in Interrupt Handler	
46	E.2.1 Saving the Memory Configuration	
46	E.2.2 Managing Main and Auxiliary Stacks	
47	E.3 User's Interrupt Handler at \$3FE	
48	E.4 Handling Break Instructions	
49	E.5 Sources of Interrupts	
50	E.6 Firmware Handling of Interrupts	
50	E.6.1 Firmware for Mouse and VBL	
52	E.6.2 Firmware for Keyboard Interrupts	
53	Using Keyboard Buffering Firmware	
53	Using Keyboard Interrupts Through Software	
54	E.6.3 Using External Interrupts Through Firmware	
55	E.6.4 Firmware for Serial Interrupts	
55	Using Serial Buffering Transparently	
56	Using Serial Interrupts Through Firmware	
57	Transmitting Serial Data	
57	A Loophole in the Firmware	
58	E.7 Bypassing the Interrupt Firmware	
58	E.7.1 Using Mouse Interrupts Without the Firmware	
59	E.7.2 Using ACIA Interrupts Without the Firmware	

61	F.1 Overview
63	F.1.1 Type of CPU
63	F.1.2 Machine Identification
64	F.2 Memory Structure
64	F.2.1 Amount and Address Ranges of RAM
65	F.2.2 Amount and Address Ranges of ROM
66	F.2.3 Peripheral-Card Memory Spaces
66	F.2.4 Hardware Addresses
67	\$C000 to \$C00F
67	\$C010 to \$C01F
68	\$C020 to \$C02F
68	\$C030 to \$C03F
68	\$C040 to \$C04F
68	\$C050 to \$C05F
69	\$C060 to \$C06F
70	\$C070 to \$C07F
70	\$C080 to \$C08F
70	\$C090 to \$C0FF
71	F.2.5 Monitors
72	F.3 I/O in General
72	F.3.1 DMA Transfers
72	F.3.2 Slots Versus Ports
72	F.3.3 Interrupts
73	F.4 Keyboard
73	F.4.1 Keys
74	F.4.2 Character Sets
75	F.5 Speaker
75	F.6 Video Display
75	F.6.1 Character Sets
76	F.6.2 MouseText
76	F.6.3 Vertical Blanking
76	F.6.4 Display Modes
77	F.7 Disk I/O
77	F.8 Serial I/O
77	F.8.1 Serial Ports Versus Serial Cards
78	F.8.2 Serial I/O Buffers
79	F.9 Mouse and Hand Controls
79	F.9.1 Mouse Input
79	F.9.2 Hand Control Input and Output
80	F.10 Cassette I/O
81	F.11 Hardware
81	F.11.1 Power
81	F.11.2 Custom Chips

<b>Appendix G</b>	<b>USA and International Models</b>	<b>83</b>
	83 G.1 Keyboard Layouts and Codes	
	85 G.1.1 USA Standard (Sholes) Keyboard	
	88 G.1.2 USA Simplified (Dvorak) Keyboard	
	89 G.1.3 ISO Layout of USA Keyboard	
	90 G.1.4 English Keyboard	
	91 G.1.5 French and Canadian Keyboards	
	93 G.1.6 German Keyboard	
	94 G.1.7 Italian Keyboard	
	96 G.1.8 Western Spanish Keyboard	
	97 G.2 ASCII Character Sets	
	99 G.3 Certifications	
	99 G.3.1 Radio Interference	
	99 G.3.2 Product Safety	
	99 G.3.3 Important Safety Instructions	
	100 G.4 Power Supply Specifications	
<b>Appendix H</b>	<b>Conversion Tables</b>	<b>103</b>
	103 H.1 Bits and Bytes	
	106 H.2 Hexadecimal and Decimal	
	107 H.3 Hexadecimal and Negative Decimal	
	109 H.4 Graphics Bits and Pieces	
	112 H.5 Peripheral Identification Numbers	
	114 H.6 Eight-Bit Code Conversions	
<b>Appendix I</b>	<b>Firmware Listings</b>	<b>125</b>

■	<i>Glossary</i>	219
■	<i>Bibliography</i>	243
■	<i>Index</i>	247
■	<i>Tell Apple Card</i>	



# *List of Figures and Tables*

## **Appendix A      *The 65C02 Microprocessor***

- 2 Table A-1    Cycle Time Differences

## **Appendix B      *Memory Map***

- 16 Table B-1    Zero Page Use  
 19 Table B-2    Page 3 Use  
 20 Table B-3    Main Memory Screen Hole Allocations  
 22 Table B-4    Auxiliary Memory Screen Hole Allocations  
 24 Table B-5    Addresses \$C000 Through \$C03F  
 25 Table B-6    Addresses \$C040 Through \$C05F  
 26 Table B-7    Addresses \$C060 Through \$C07F  
 27 Table B-8    Addresses \$C080 Through \$C0AF  
 28 Table B-9    Addresses \$C0B0 Through \$C0FF

## **Appendix C      *Important Firmware Locations***

- 32 Table C-1    Serial Port 1 Addresses  
 32 Table C-2    Serial Port 2 Addresses  
 33 Table C-3    Video Firmware Addresses  
 33 Table C-4    Mouse Port Addresses  
 34 Table C-5    Apple IIc Enhanced Video  
                     and Miscellaneous Firmware  
 35 Table C-6    Apple IIc Monitor Entry Points  
                     and Vectors

---

**Appendix E****Interrupts**

- 45 Figure E-1 Interrupt-Handling Sequence
- 58 Table E-1 Activating Mouse Interrupts
- 58 Table E-2 Reading Mouse Interrupts

---

**Appendix F****Apple II Series Differences**

- 80 Figure F-1 Apple II, II Plus, and IIe Hand Control Signals

---

**Appendix G****USA and International Models**

- 85 Figure G-1 USA Standard or *Sholes* Keyboard (Keyboard Switch Up)
- 86 Table G-1 Keys and ASCII Codes
- 88 Figure G-2 USA Simplified or *Dvorak* Keyboard (Keyboard Switch Down)
- 89 Figure G-3 ISO Version of USA Standard Keyboard (Keyboard Switch Up)
- 90 Figure G-4 English Keyboard (Keyboard Switch Down)
- 90 Table G-2 English Keyboard Code Differences From Table G-1
- 91 Figure G-5 French Keyboard (Keyboard Switch Down)
- 92 Figure G-6 Canadian Keyboard (Keyboard Switch Down)
- 92 Table G-3 French and Canadian Keyboard Code Differences From Table G-1
- 93 Figure G-7 German Keyboard (Keyboard Switch Down)
- 93 Table G-4 German Keyboard Code Differences From Table G-1
- 94 Figure G-8 Italian Keyboard (Keyboard Switch Down)
- 95 Table G-5 Italian Keyboard Code Differences From Table G-1
- 96 Figure G-9 Western Spanish Keyboard (Keyboard Switch Down)
- 96 Table G-6 Western Spanish Keyboard Code Differences From Table G-1
- 98 Table G-7 ASCII Code Equivalents
- 100 Table G-8 50 Hz Power Supply Specifications

## **Appendix H**

### **Conversion Tables**

104	Table H-1	What a Bit Can Represent
105	Figure H-1	Bits, Nibbles, and Bytes
106	Table H-2	Hexadecimal/Decimal Conversion
108	Table H-3	Decimal to Negative Decimal Conversion
109	Table H-4	Hexadecimal Values for High-Resolution Dot Patterns
113	Table H-5	PIN Numbers
115	Table H-6	Control Characters, High Bit Off
116	Table H-7	Special Characters, High Bit Off
117	Table H-8	Uppercase Characters, High Bit Off
118	Table H-9	Lowercase Characters, High Bit Off
119	Table H-10	Control Characters, High Bit On
120	Table H-11	Special Characters, High Bit On
121	Table H-12	Uppercase Characters, High Bit On
122	Table H-13	Lowercase Characters, High Bit On

# *Preface*



This volume, Volume 2 of the *Apple IIc Reference Manual*, contains nine appendixes, a bibliography, and a glossary.

Appendix A contains a description of the differences between the 6502 and the 65C02 microprocessors, plus a reprint of the manufacturer's data sheet for the 65C02 microprocessor.

Appendixes B and C contain tables of the important RAM, ROM, and hardware addresses in the Apple IIc. The reader can use these tables to find locations by address, the index to find them by label, the firmware listings to find them as defined and used, and the chapters to find them described in the context of their function.

Appendix B is a memory map of the Apple IIc, including detailed tables of page zero, page three, the screen holes, and the hardware page.

Appendix C lists the *published* firmware entry points, arranged by address, and indicates where in the manual they are described. The list includes I/O firmware (pages \$C1 through \$CF) and Monitor firmware (pages \$F0 through \$FF). For Applesoft interpreter firmware (pages \$D0 through \$EF), refer to the *Applesoft BASIC Programmer's Reference Manual*, Volumes 1 and 2.

Appendix D discusses what operating systems and languages run on the Apple IIc, and what features they do and do not use.

Appendix E describes how to use the Apple IIc's interrupt handling capabilities.

Appendix F contains an overview of the differences among the Apple II series computers.

---

Appendix G contains the keyboard layouts, code conversion tables, and external power supply characteristics of USA and international models of the Apple IIc.

Appendix H contains reference tables for code and number base conversion.

Appendix I contains a listing of the source code for the Monitor, enhanced video firmware, and input/output firmware contained in the Apple IIc. The listings do not include the built-in Applesoft interpreter, which is discussed in the *Applesoft BASIC Programmer's Reference Manual*.

The Bibliography lists articles and books containing additional information about the Apple IIc and related products.

The Glossary defines many of the technical terms used in this manual.



# *The 65C02 Microprocessor*



This appendix contains a description of the differences between the 6502 and the 65C02 microprocessor. It also contains the data sheet for the NCR 65C02 microprocessor.

In the data sheet tables, execution times are specified in number of cycles. One cycle time for the Apple IIc equals 0.978 microseconds.

If you want to write programs that execute on all computers in the Apple II series, make sure your code uses only the subset of instructions present on the 6502.



## A.1 Differences Between 6502 and 65C02

The data sheet lists the new instructions and addressing modes of the 65C02. This section supplements that information by listing the instructions whose execution times or results have changed.

### A.1.1 Differing Cycle Times

In general, differences in execution times are significant only in time-dependent code, such as precise wait loops. Fortunately, instructions with changed execution times are few.

Table A-1 lists the instructions whose number of instruction execution cycles on the 65C02 is different from the number on the 6502.

*Table A-1. Cycle Time Differences*

Instruction/Mode	Opcode	6502 Cycles	65C02 Cycles
ASL Absolute, X	1E	7	6
DEC Absolute, X	DE	7	6
INC Absolute, X	FE	7	6
JMP (Absolute)	6C	5	6
LSR Absolute, X	5E	7	6
ROL Absolute, X	3E	7	6
ROR Absolute, X	7E	7	6

---

### ***A.1.2 Differing Instruction Results***

It is important to note that the BIT instruction when used in immediate mode (code \$89) leaves Processor Status Register bits 7 (N) and 6 (V) unchanged on the 65C02. On the 6502, all modes of the BIT instruction have the same effect on the Status Register: the value of memory bit 7 is placed in status bit 7, and memory bit 6 is placed in status bit 6. However, all BIT instructions on both versions of the processor set status bit 1 (Z) if the memory location contained a zero.

Also note that if the JMP indirect instruction (code \$6C) references an indirect address location that spans a page boundary, the 65C02 fetches the high-order byte of the effective address from the first byte of the next page, while the 6502 fetches it from the first byte of the current page. For example, JMP (\$2FF) gets ADL from location \$2FF on both processors. But on the 65C02, ADH comes from \$300; on the 6502, ADH comes from \$200.

---

## **A.2 Data Sheet**

The remaining pages of this appendix are copyright 1982, NCR Corporation, Dayton, Ohio, and are reprinted with their permission.

#### ■ GENERAL DESCRIPTION

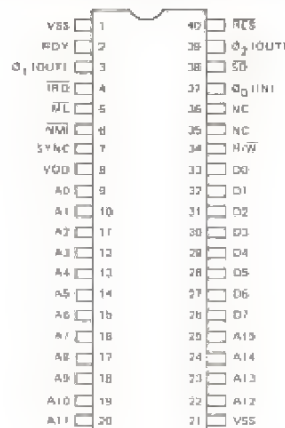
The NCR CMOS 6502 is an 8-bit microprocessor which is software compatible with the NMOS 6502. The NCR6502 hardware interfaces with all 6500 peripherals. The enhancements include ten additional instructions, expanded operational codes and two new addressing modes. This microprocessor has all of the advantages of CMOS technology: low power consumption, increased noise immunity and higher reliability. The CMOS 6502 is a low power high performance microprocessor with applications in the consumer, business, automotive and communications market.

## ■ FEATURES

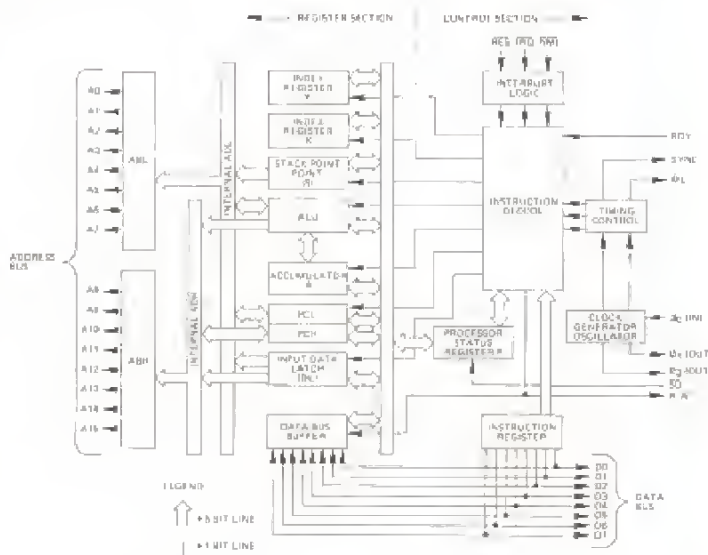
- Enhanced software performance including 27 additional OP codes encompassing ten new instructions and two additional addressing modes.
- 66 microprocessor instructions.
- 15 addressing modes.
- 178 operational codes.
- 1MHz, 2MHz operation.
- Operates at frequencies as low as 200 Hz for even lower power consumption (pseudo-static: stop during  $\Phi_2$  high).
- Compatible with NMOS 6500 series microprocessors.
- 64 K-byte addressable memory.
- Interrupt capability.
- Lower power consumption.  
4mA @ 1MHz.
- +5 volt power supply.
- 8-bit bidirectional data bus.
- Bus Compatible with M6800.
- Non-maskable interrupt.
- 40 pin dual-in-line packaging.
- 8-bit parallel processing
- Decimal and binary arithmetic.
- Pipeline architecture.
- Programmable stack pointer.
- Variable length stack.
- Optional internal pullups for (RDY, IRQ,  $\overline{SC}$ , NMI and RES)

\* Specifications are subject to change without notice.

- **PIN CONFIGURATION**



### ▪ NCR65C02 BLOCK DIAGRAM



# NCR65C02

## ABSOLUTE MAXIMUM RATINGS: ( $V_{DD} = 5.0\text{ V} \pm 5\%$ , $V_{SS} = 0\text{ V}$ , $T_A = 0^\circ\text{C to } 70^\circ\text{C}$ )

RATING	SYMBOL	VALUE	UNIT
SUPPLY VOLTAGE	$V_{DD}$	-0.3 to +7.0	V
INPUT VOLTAGE	$V_{IN}$	-0.3 to +7.0	V
OPERATING TEMP.	$T_A$	0 to +70	$^\circ\text{C}$
STORAGE TEMP.	$T_{STG}$	-55 to +150	$^\circ\text{C}$

## PIN FUNCTION

PIN	FUNCTION
A0 - A15	Address Bus
D0 - D7	Data Bus
IRQ *	Interrupt Request
RDY *	Ready
ML	Memory Lock
NMI *	Non-Maskable Interrupt
SYNC	Synchronize
RES *	Reset
SO *	Set Overflow
NC	No Connection
R/W	Read/Write
VDD	Power Supply (+5V)
VSS	Internal Logic Ground
$\phi_0$	Clock Input
$\phi_1, \phi_2$	Clock Output

\*This pin has an optional internal pullup for a No Connect condition.

## DC CHARACTERISTICS

	SYMBOL	MIN.	TYP.	MAX.	UNIT
Input High Voltage $\phi_0$ (IN)	$V_{IH}$	$V_{SS} + 2.4$	—	$V_{DD}$	V
Input High Voltage RES, NMI, RDY, IRQ, Data, S.O.		$V_{SS} + 2.0$	—	—	V
Input Low Voltage $\phi_0$ (IN)	$V_{IL}$	$V_{SS} - 0.3$	—	$V_{SS} + 0.4$	V
Input Low Voltage RES, NMI, RDY, IRQ, Data, S.O.		—	—	$V_{SS} + 0.8$	V
Input Leakage Current ( $V_{IN} = 0$ to 5.25V, $V_{DD} = 5.25\text{V}$ )	$I_{IN}$	-30	—	+30	$\mu\text{A}$
Without pullups		—	—	+1.0	$\mu\text{A}$
Three State (Off State) Input Current ( $V_{IN} = 0.4$ to 2.4V, $V_{CC} = 5.25\text{V}$ )		—	—	10	$\mu\text{A}$
Data Lines	$I_{TSI}$	—	—	10	$\mu\text{A}$
Output High Voltage ( $I_{OH} = -100\text{ }\mu\text{A}$ , $V_{DD} = 4.75\text{V}$ SYNC, Data, A0-A15, R/W)	$V_{OH}$	$V_{SS} + 2.4$	—	—	V
Out Low Voltage ( $I_{OL} = 1.6\text{mA}$ , $V_{DD} = 4.75\text{V}$ SYNC, Data, A0-A15, R/W)	$V_{OL}$	—	—	$V_{SS} + 0.4$	V
Supply Current $f = 1\text{MHz}$	$I_{DD}$	—	—	4	mA
Supply Current $f = 2\text{MHz}$	$I_{DD}$	—	—	8	mA
Capacitance ( $V_{IN} = 0$ , $T_A = 25^\circ\text{C}$ , $f = 1\text{MHz}$ )	C	—	—	—	pF
Logic	$C_{IN}$	—	—	5	
Data		—	—	10	
A0-A15, R/W, SYNC	$C_{out}$	—	—	10	
$\phi_0$ (IN)	$C_{\phi_0}$	—	—	10	



# NCR65C02

## ■ AC CHARACTERISTICS $V_{DD} = 5.0V \pm 5\%$ , $T_A = 0^\circ C$ to $70^\circ C$ , Load = 1 TTL + 130 pF

Parameter	Symbol	1MHz		2MHz		3MHz		Unit
		Min	Max	Min	Max	Min	Max	
Delay Time, $\theta_0$ (IN) to $\theta_2$ (OUT)	$t_{DLY}$	—	60	—	60	20	60	nS
Delay Time, $\theta_1$ (OUT) to $\theta_2$ (OUT)	$t_{DLY1}$	—20	20	—20	20	—20	20	nS
Cycle Time	$t_{CYC}$	1.0	5000*	0.50	5000*	0.33	5000*	$\mu S$
Clock Pulse Width Low	$t_{PL}$	460	—	220	—	160	—	nS
Clock Pulse Width High	$t_{PH}$	460	—	220	—	160	—	nS
Fall Time, Rise Time	$t_F, t_R$	—	25	—	25	—	25	nS
Address Hold Time	$t_{AH}$	20	—	20	—	0	—	nS
Address Setup Time	$t_{AS}$	—	225	—	140	—	110	nS
Access Time	$t_{ACC}$	650	—	310	—	170	—	nS
Read Data Hold Time	$t_{DHR}$	10	—	10	—	10	—	nS
Read Data Setup Time	$t_{DSU}$	100	—	60	—	60	—	nS
Write Data Delay Time	$t_{MDS}$	—	30	—	30	—	30	nS
Write Data Hold Time	$t_{DHW}$	20	—	20	—	15	—	nS
SO Setup Time	$t_{SO}$	100	—	100	—	100	—	nS
Processor Control Setup Time**	$t_{PCS}$	200	—	150	—	150	—	nS
SYNC Setup Time	$t_{SYNC}$	—	225	—	140	—	100	nS
ML Setup Time	$t_{ML}$	—	225	—	140	—	100	nS
Input Clock Rise/Fall Time	$t_{F0}, t_{R0}$	—	25	—	25	—	25	nS

\*NCR65C02 can be held static with  $\theta_2$  high.

\*\*This parameter must only be met to guarantee that the signal will be recognized at the current clock cycle.

## ■ MICROPROCESSOR OPERATIONAL ENHANCEMENTS

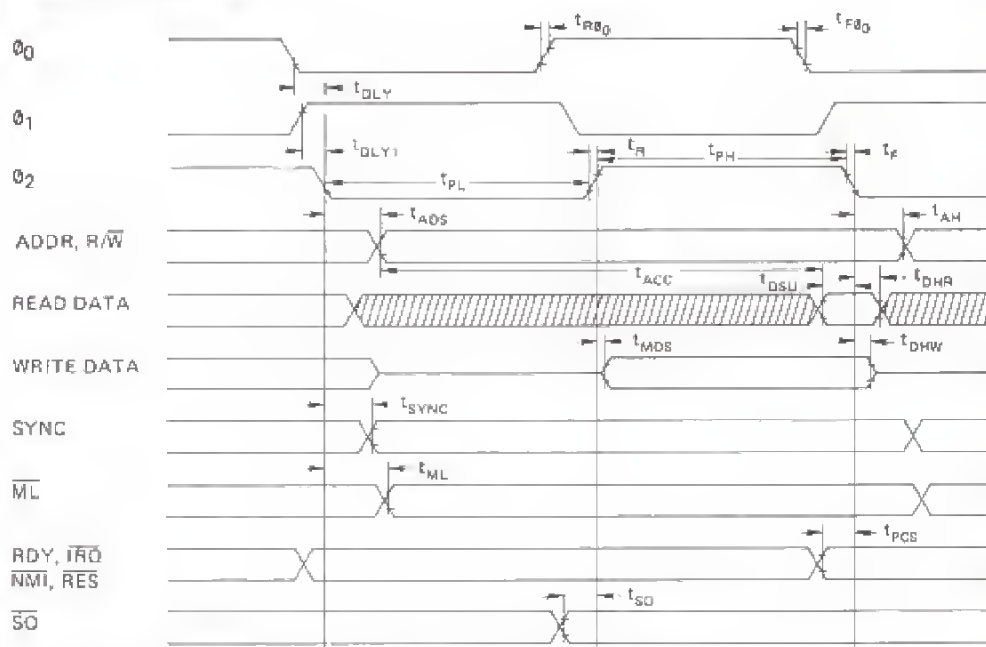
Function	NMOS 6502 Microprocessor	NCR65C02 Microprocessor
Indexed addressing across page boundary.	Extra read of invalid address.	Extra read of last instruction byte.
Execution of invalid op codes.	Some terminate only by reset. Results are undefined.	All are NOPs (reserved for future use). Op Code      Bytes      Cycles X2              2              2 X3, X7, XB, XF      1              1 44              2              3 54, D4, F4              2              4 5C              3              8 DC, FC              3              4
Jump indirect, operand = XXFF.	Page address does not increment.	Page address increments and adds one additional cycle.
Read/modify/write instructions at effective address.	One read and two write cycles.	Two read and one write cycle.
Decimal flag.	Indeterminate after reset.	Initialized to binary mode (D=0) after reset and interrupts.
Flags after decimal operation.	Invalid N, V and Z flags.	Valid flag adds one additional cycle.
Interrupt after fetch of BRK instruction.	Interrupt vector is loaded, BRK vector is ignored.	BRK is executed, then interrupt is executed.

## ■ MICROPROCESSOR HARDWARE ENHANCEMENTS

Function	NMOS 6502	NCR65C02
Assertion of Ready RDY during write operations.	Ignored.	Stops processor during $\theta_2$ .
Unused input-only pins ( $\overline{IRQ}$ , $\overline{NMI}$ , RDY, RES, SO).	Must be connected to low impedance signal to avoid noise problems.	Connected internally by a high-resistance to $V_{DD}$ (approximately 250 K ohm.)

## NCR65C02

### ■ TIMING DIAGRAM



Note: All timing is referenced from a high voltage of 2.0 volts and a low voltage of 0.8 volts.

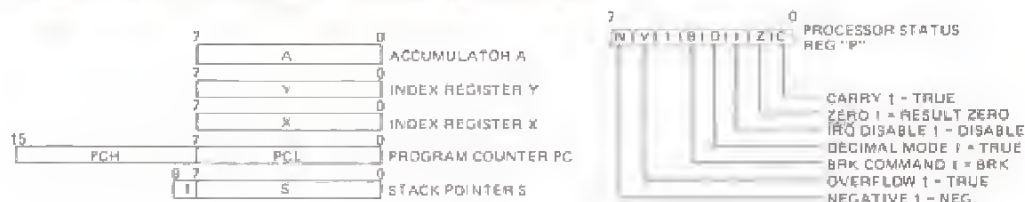
### ■ NEW INSTRUCTION MNEMONICS

HEX	MNEMONIC	DESCRIPTION
80	BRA	Branch relative always [Relative]
3A	DEA	Decrement accumulator [Accum]
1A	INA	Increment accumulator [Accum]
DA	PHX	Push X on stack [Implied]
5A	PHY	Push Y on stack [Implied]
FA	PLX	Pull X from stack [Implied]
7A	PLY	Pull Y from stack [Implied]
9C	STZ	Store zero [Absolute]
9E	STZ	Store zero [ABS, X]
64	STZ	Store zero [Zero page]
74	STZ	Store zero [ZPG, X]
1C	TRB	Test and reset memory bits with accumulator [Absolute]
14	TRB	Test and reset memory bits with accumulator [Zero page]
0C	TSB	Test and set memory bits with accumulator [Absolute]
04	TSB	Test and set memory bits with accumulator [Zero page]

### ■ ADDITIONAL INSTRUCTION ADDRESSING MODES

HEX	MNEMONIC	DESCRIPTION
72	ADC	Add memory to accumulator with carry [(ZPG)]
32	AND	"AND" memory with accumulator [(ZPG)]
3C	BIT	Test memory bits with accumulator [ABS, X]
34	BIT	Test memory bits with accumulator [ZPG, X]
D2	CMP	Compare memory and accumulator [(ZPG)]
52	EOR	"Exclusive Or" memory with accumulator [(ZPG)]
7C	JMP	Jump (New addressing mode) [ABS(IND, X)]
B2	LDA	Load accumulator with memory [(ZPG)]
12	ORA	"OR" memory with accumulator [(ZPG)]
F2	SBC	Subtract memory from accumulator with borrow [(ZPG)]
92	STA	Store accumulator in memory [(ZPG)]

## ■ MICROPROCESSOR PROGRAMMING MODEL



## ■ FUNCTIONAL DESCRIPTION

### Timing Control

The timing control unit keeps track of the instruction cycle being monitored. The unit is set to zero each time an instruction fetch is executed and is advanced at the beginning of each phase one clock pulse for as many cycles as is required to complete the instruction. Each data transfer which takes place between the registers depends upon decoding the contents of both the instruction register and the timing control unit.

### Program Counter

The 16-bit program counter provides the addresses which step the microprocessor through sequential instructions in a program.

Each time the microprocessor fetches an instruction from program memory, the lower byte of the program counter (PCL) is placed on the low-order bits of the address bus and the higher byte of the program counter (PCH) is placed on the high-order 8 bits. The counter is incremented each time an instruction or data is fetched from program memory.

### Instruction Register and Decode

Instructions fetched from memory are gated onto the internal data bus. These instructions are latched into the instruction register, then decoded, along with timing and interrupt signals, to generate control signals for the various registers.

### Arithmetic and Logic Unit (ALU)

All arithmetic and logic operations take place in the ALU including incrementing and decrementing internal registers (except the program counter). The ALU has no internal memory and is used only to perform logical and transient numerical operations.

### Accumulator

The accumulator is a general purpose 8-bit register that stores the results of most arithmetic and logic operations, and in addition, the accumulator usually contains one of the two data words used in these operations.

### Index Registers

There are two 8-bit index registers (X and Y), which may be used to count program steps or to provide an index value to be used in generating an effective address.

When executing an instruction which specifies indexed addressing, the CPU fetches the op code and the base address, and modifies the address by adding the index register to it prior to performing the desired operation. Pre- or post-indexing of indirect addresses is possible (see addressing modes).

### Stack Pointer

The stack pointer is an 8-bit register used to control the addressing of the variable-length stack on page one. The stack pointer is automatically incremented and decremented under control of the microprocessor to perform stack manipulations under direction of either the program or interrupts (NMI and IRQ). The stack allows simple implementation of nested subroutines and multiple level interrupts. The stack pointer should be initialized before any interrupts or stack operations occur.

### Processor Status Register

The 8-bit processor status register contains seven status flags. Some of the flags are controlled by the program, others may be controlled both by the program and the CPU. The 6500 instruction set contains a number of conditional branch instructions which are designed to allow testing of these flags (see microprocessor programming model).

## NCR65C02

### ■ ADDRESSING MODES

Fifteen addressing modes are available to the user of the NCR65C02 microprocessor. The addressing modes are described in the following paragraphs:

**Implied Addressing [Implied]**  
In the implied addressing mode, the address containing the operand is implicitly stated in the operation code of the instruction.

**Accumulator Addressing [Accum]**  
This form of addressing is represented with a one byte instruction and implies an operation on the accumulator.

**Immediate Addressing [Immediate]**  
With immediate addressing, the operand is contained in the second byte of the instruction; no further memory addressing is required.

**Absolute Addressing [Absolute]**  
For absolute addressing, the second byte of the instruction specifies the eight low-order bits of the effective address, while the third byte specifies the eight high-order bits. Therefore, this addressing mode allows access to the total 64K bytes of addressable memory.

**Zero Page Addressing [Zero Page]**  
Zero page addressing allows shorter code and execution times by only fetching the second byte of the instruction and assuming a zero high address byte. The careless use of zero page addressing can result in significant increase in code efficiency.

**Absolute Indexed Addressing [ABS, X or ABS, Y]**  
Absolute indexed addressing is used in conjunction with X or Y index register and is referred to as "Absolute, X," and "Absolute, Y." The effective address is formed by adding the contents of X or Y to the address contained in the second and third bytes of the instruction. This mode allows the index register to contain the index or count value and the instruction to contain the base address. This type of indexing allows any location referencing and the index to modify multiple fields, resulting in reduced coding and execution time.

**Zero Page Indexed Addressing [ZPG, X or ZPG, Y]**  
Zero page absolute addressing is used in conjunction with the index register and is referred to as "Zero Page, X" or "Zero Page, Y." The effective address is calculated by adding the second byte to the contents of the index register. Since this is a form of "Zero Page" addressing, the content of the second byte references a location in page zero. Additionally, due to the "Zero Page" addressing nature of this mode, no carry is added to the high-order eight bits of memory, and crossing of page boundaries does not occur.

**Relative Addressing [Relative]**  
Relative addressing is used only with branch instructions;

it establishes a destination for the conditional branch. The second byte of the instruction becomes the operand which is an "Offset" added to the contents of the program counter when the counter is set at the next instruction. The range of the offset is -128 to +127 bytes from the next instruction.

**Zero Page Indexed Indirect Addressing [(IND, X)]**  
With zero page indexed indirect addressing (usually referred to as indirect X) the second byte of the instruction is added to the contents of the X index register; the carry is discarded. The result of this addition points to a memory location on page zero whose contents is the low-order eight bits of the effective address. The next memory location in page zero contains the high-order eight bits of the effective address. Both memory locations specifying the high- and low-order bytes of the effective address must be in page zero.

**\*Absolute Indexed Indirect Addressing [ABS(IND, X)] (Jump Instruction Only)**  
With absolute indexed indirect addressing the contents of the second and third instruction bytes are added to the X register. The result of this addition, points to a memory location containing the lower-order eight bits of the effective address. The next memory location contains the higher-order eight bits of the effective address.

**Indirect Indexed Addressing [(IND), Y]**  
This form of addressing is usually referred to as indirect, Y. The second byte of the instruction points to a memory location in page zero. The contents of this memory location are added to the contents of the Y index register, the result being the low-order eight bits of the effective address. The carry from this addition is added to the contents of the next page zero memory location, the result being the high-order eight bits of the effective address.

**\*Zero Page Indirect Addressing [(ZPG)]**  
In the zero page indirect addressing mode, the second byte of the instruction points to a memory location on page zero containing the low-order byte of the effective address. The next location on page zero contains the high-order byte of the effective address.

**Absolute Indirect Addressing [(ABS)] (Jump Instruction Only)**  
The second byte of the instruction contains the low-order eight bits of a memory location. The high-order eight bits of that memory location is contained in the third byte of the instruction. The contents of the fully specified memory location is the low-order byte of the effective address. The next memory location contains the high-order byte of the effective address which is loaded into the 16 bit program counter.

NOTE: \* = New Address Modes



## ■ SIGNAL DESCRIPTION

### Address Bus (A0-A15)

A0-A15 forms a 16-bit address bus for memory and I/O exchanges on the data bus. The output of each address line is TTL compatible, capable of driving one standard TTL load and 130pF.

### Clocks ( $\Phi_0$ , $\Phi_1$ , and $\Phi_2$ )

$\Phi_0$  is a TTL level input that is used to generate the internal clocks in the 6502. Two full level output clocks are generated by the 6502. The  $\Phi_2$  clock output is in phase with  $\Phi_0$ . The  $\Phi_1$  output pin is 180° out of phase with  $\Phi_0$ . (See timing diagram.)

### Data Bus (D0-D7)

The data lines (D0-D7) constitute an 8-bit bidirectional data bus used for data exchanges to and from the device and peripherals. The outputs are three-state buffers capable of driving one TTL load and 130 pF.

### Interrupt Request ( $\overline{\text{IRQ}}$ )

This TTL compatible input requests that an interrupt sequence begin within the microprocessor. The  $\overline{\text{IRQ}}$  is sampled during  $\Phi_2$  operation; if the interrupt flag in the processor status register is zero, the current instruction is completed and the interrupt sequence begins during  $\Phi_1$ . The program counter and processor status register are stored in the stack. The microprocessor will then set the interrupt mask flag high so that no further IRQs may occur. At the end of this cycle, the program counter low will be loaded from address FFFE, and program counter high from location FFFF, transferring program control to the memory vector located at these addresses. The RDY signal must be in the high state for any interrupt to be recognized. A 3K ohm external resistor should be used for proper wire OR operation.

### Memory Lock ( $\overline{\text{ML}}$ )

In a multiprocessor system, the  $\overline{\text{ML}}$  output indicates the need to defer the arbitration of the next bus cycle to ensure the integrity of read-modify-write instructions.  $\overline{\text{ML}}$  goes low during ASL, DEC, INC, LSR, ROL, ROR, TRB, TSB memory referencing instructions. This signal is low for the modify and write cycles.

### Non-Maskable Interrupt ( $\overline{\text{NMI}}$ )

A negative-going edge on this input requests that a non-maskable interrupt sequence be generated within the microprocessor. The  $\overline{\text{NMI}}$  is sampled during  $\Phi_2$ ; the current instruction is completed and the interrupt sequence begins during  $\Phi_1$ . The program counter is loaded with the interrupt vector from locations FFFA (low byte) and FFFB (high byte), thereby transferring program control to the non-maskable interrupt routine.

Note: Since this interrupt is non-maskable, another  $\overline{\text{NMI}}$  can occur before the first is finished. Care should be taken when using  $\overline{\text{NMI}}$  to avoid this.

### Ready (RDY)

This input allows the user to single-cycle the microprocessor on all cycles including write cycles. A negative transition to the low state, during or coincident with phase one ( $\Phi_1$ ), will halt the microprocessor with the output address lines reflecting the current address being fetched. This condition will remain through a subsequent phase two ( $\Phi_2$ ) in which the ready signal is low. This feature allows microprocessor interfacing with low-speed memory as well as direct memory access (DMA).

### Reset ( $\overline{\text{RES}}$ )

This input is used to reset the microprocessor. Reset must be held low for at least two clock cycles after VDD reaches operating voltage from a power down. A positive transition on this pin will then cause an initialization sequence to begin. Likewise, after the system has been operating, a low on this line of at least two cycles will cease microprocessing activity, followed by initialization after the positive edge on  $\overline{\text{RES}}$ .

When a positive edge is detected, there is an initialization sequence lasting six clock cycles. Then the interrupt mask flag is set, the decimal mode is cleared, and the program counter is loaded with the restart vector from locations FFFC (low byte) and FFFD (high byte). This is the start location for program control. This input should be high in normal operation.

### Read/Write ( $\overline{\text{R/W}}$ )

This signal is normally in the high state indicating that the microprocessor is reading data from memory or I/O bus. In the low state the data bus has valid data from the microprocessor to be stored at the addressed memory location.

### Set Overflow ( $\overline{\text{SO}}$ )

A negative transition on this line sets the overflow bit in the status code register. The signal is sampled on the trailing edge of  $\Phi_1$ .

### Synchronize (SYNC)

This output line is provided to identify those cycles during which the microprocessor is doing an OP CODE fetch. The SYNC line goes high during  $\Phi_1$  of an OP CODE fetch and stays high for the remainder of that cycle. If the RDY line is pulled low during the  $\Phi_1$  clock pulse in which SYNC went high, the processor will stop in its current state and will remain in the state until the RDY line goes high. In this manner, the SYNC signal can be used to control RDY to cause single instruction execution.

# NCR65C02

## INSTRUCTION SET — ALPHABETICAL SEQUENCE

ADC	Add Memory to Accumulator with Carry	LDX	Load Index X with Memory
AND	"AND" Memory with Accumulator	LDY	Load Index Y with Memory
ASL	Shift One Bit Left	LSR	Shift One Bit Right
BCC	Branch on Carry Clear	NOP	No Operation
BCS	Branch on Carry Set	ORA	"OR" Memory with Accumulator
BEQ	Branch on Result Zero	PHA	Push Accumulator on Stack
BIT	Test Memory Bits with Accumulator	PHP	Push Processor Status on Stack
BMI	Branch on Result Minus	*PHK	Push Index X on Stack
BNE	Branch on Result not Zero	*PHY	Push Index Y on Stack
BPL	Branch on Result Plus	PLA	Pull Accumulator from Stack
*BRA	Branch Always	PLP	Pull Processor Status from Stack
BRK	Force Break	*PLX	Pull Index X from Stack
BVC	Branch on Overflow Clear	*PLY	Pull Index Y from Stack
BVS	Branch on Overflow Set	ROL	Rotate One Bit Left
CLC	Clear Carry Flag	ROR	Rotate One Bit Right
CLE	Clear Decimal Mode	RTI	Return from Interrupt
CLI	Clear Interrupt Disable Bit	RTS	Return from Subroutine
CLV	Clear Overflow Flag	SBC	Subtract Memory from Accumulator with Borrow
CMP	Compare Memory and Accumulator	SEC	Set Carry Flag
CPX	Compare Memory and Index X	SED	Set Decimal Mode
CPY	Compare Memory and Index Y	SEI	Set Interrupt Disable Bit
*DEA	Decrement Accumulator	STP	Store Accumulator in Memory
DEC	Decrement by One	STX	Store Index X in Memory
DEX	Decrement Index X by One	STY	Store Index Y in Memory
DEY	Decrement Index Y by One	*STZ	Store Zero in Memory
EOR	"Exclusive-or" Memory with Accumulator	TAX	Transfer Accumulator to Index X
*INA	Increment Accumulator	TAY	Transfer Accumulator to Index Y
INC	Increment by One	*TRB	Test and Reset Memory Bits with Accumulator
INX	Increment Index X by One	TSB	Test and Set Memory Bits with Accumulator
INY	Increment Index Y by One	TSX	Transfer Stack Pointer to Index X
JMP	Jump to New Location	TXA	Transfer Index X to Accumulator
JSR	Jump to New Location Saving Return Address	TXS	Transfer Index X to Stack Pointer
LDA	Load Accumulator with Memory	TYA	Transfer Index Y to Accumulator

Note: \* = New Instruction

## MICROPROCESSOR OP CODE TABLE

S	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	BRK rel	ORA ind, X			TSB* zpg	ORA zpg	ASL zpg		PHP	ORA imm	ASL A		TSB* abs	ORA abs	ASL abs		0
1	BPL rel	ORA ind, Y	ORA*† zpg		TRB* zpg	ORA zpg, X	ASL zpg, X		CLC	ORA abs, Y	INA* A		TRB* abs	ORA abs, X	ASL abs, X		1
2	JSR abs	AND ind, X			BIT zpg	AND zpg	ROL zpg		PLP	AND imm	ROL A		BIT abs	AND abs	ROL abs		2
3	BMI rel	AND ind, Y	AND*† zpg		BIT* zpg, X	AND zpg, X	ROL zpg, X		SEC	AND abs, Y	DEA* A		BIT*† abs, X	AND abs, X	ROL abs, X		3
4	RTI ind, X	EOR ind, X			EOR zpg	LSR zpg, X			PHA	EOR imm	LSR A		JMP abs	EOR abs	LSR abs		4
5	BVC rel	EOR ind, Y	EOR*† zpg		EOR zpg, X	LSR zpg, X			CLI	EOR abs, Y	PHY* A			EOR abs, X	LSR abs, X		5
6	RTS rel	ADC ind, X			STZ* zpg	ADC zpg	ROR zpg		PLA	ADC imm	ROR A		JMP (abs)	ADC abs	ROR abs		6
7	BVS rel	ADC ind, Y	ADC*† zpg		STZ* zpg, X	ADC zpg, X	ROR zpg, X		SEI	ADC abs, Y	PLY* A		JMP*† abs, ind, X	ADC abs, X	ROR abs, X		7
8	BRA* rel	STA ind, X			STY zpg	STA zpg	STX zpg		DEY	BIT* imm	TXA		STY abs	STA abs	STX abs		8
9	BCC rel	STA ind, Y	STA*† zpg		STY zpg, X	STA zpg, X	STX zpg, Y		TYA	STA abs, Y	TXS		STZ* abs	STA abs, X	STZ* abs, X		9
A	LDY imm	LDA ind, X	LDX imm		LDY zpg	LDA zpg	LDX zpg		TAY	LDA imm	TAX		LDY abs	LDA abs	LDX abs		A
B	BCS rel	LDA ind, Y	LDA*† zpg		LDY zpg, X	LDA zpg, X	LDX zpg, Y		CLV	LDA abs, Y	TSX		LDY abs, X	LDA abs, X	LDX abs, Y		B
C	CPY imm	CMP ind, X			CPY zpg	CMP zpg	DEC zpg		INY	CMP imm	DEX		CPY abs	CMP abs	DEC abs		C
D	BNE rel	CMP ind, Y	CMP*† zpg			CMP zpg, X	DEC zpg, X		CLD	CMP abs, Y	PHX* A			CMP abs, X	DEC abs, X		D
E	CPX imm	SBC ind, X			CPX zpg	SBC zpg	INC zpg		INX	SBC imm	NOP		CPX abs	SBC abs	INC abs		E
F	BEQ rel	SBC ind, Y	SBC*† zpg			SBC zpg, X	INC zpg, X		SED	SBC abs, Y	PLX* A			SBC abs, X	INC abs, X		F
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

Note: \* = New OP Codes

Note: † = New Address Modes

### ■ OPERATIONAL CODES, EXECUTION TIME, AND MEMORY REQUIREMENTS

[illegible]

Notes:

1. Add 1 to "n" if page boundary is crossed.
2. Add 1 to "n" if branch occurs to same page  
Add 2 to "n" if branch occurs to different page.
3. Add 1 to "n" if decimal mode.
4. V bit equals memory bit 6 prior to execution  
N bit equals memory bit 7 prior to execution

X index X

$$Y = \text{Index } Y$$

A. Accruals of

M Memory per effective address

285 Memory ptr stack pointer

+ Add

— Subtract

And

Y Or

☒ Exclusive or

n No. Cycles

\* No Bytes

Memory bit 6

Mr. Thompson

## REFERENCES

\*5. The immediate addressing mode of the BIT instruction leaves bits 6 & 7 (V & N) in the Processor Status Code Register unchanged.



# *Memory Map*

Appendix H explains the general rules and tables for converting numbers from one of these forms to another. For memory map diagrams, refer to Chapter 2. Figure 2-2 is an overall memory map, Figure 2-3 is a map of bank-switched memory, and Figure 2-11 is a map of the 48K memory space.

This appendix lists all important RAM and hardware locations in address order and describes them briefly. It also provides cross-references to the section of the manual where they are described further. Appendix C contains a similar list for important firmware addresses.

The tables in this appendix list addresses in either two or three forms: the hexadecimal form (preceded by a dollar sign) for use in assembly language; the decimal form for use in Applesoft BASIC; and (for numbers greater than 32767) the complementary decimal value for use in Apple Integer BASIC.

## B.1 Page Zero

For Monitor zero page usage, refer to the firmware listings. For zero page use by the languages and operating systems, refer to the appropriate reference manuals.

Table B-1 lists the zero page addresses in hexadecimal and decimal form, followed by symbols denoting the firmware or system software that uses them.

- M denotes the Monitor.
- A denotes Applesoft BASIC.
- I denotes Integer BASIC.
- D denotes DOS.
- P denotes ProDOS. Locations whose contents ProDOS saves and restores afterward have a P in parentheses, indicating that ProDOS has no net effect on them.

Table B-1. Zero Page Use

Hex	Dec	Used by	Hex	Dec	Used by
\$00	0	A	\$30	48	M
\$01	1	A	\$31	49	M
\$02	2	A	\$32	50	M
\$03	3	A	\$33	51	M
\$04	4	A	\$34	52	M
\$05	5	A	\$35	53	M
\$06	6		\$36	54	M
\$07	7		\$37	55	M
\$08	8		\$38	56	M
\$09	9		\$39	57	M
\$0A	10	A	\$3A	58	M
\$0B	11	A	\$3B	59	M
\$0C	12	A	\$3C	60	M
\$0D	13	A	\$3D	61	M
\$0E	14	A	\$3E	62	M
\$0F	15	A	\$3F	63	M
\$10	16	A	\$40	64	M
\$11	17	A	\$41	65	M
\$12	18	A	\$42	66	M
\$13	19	A	\$43	67	M
\$14	20	A	\$44	68	M
\$15	21	A	\$45	69	M
\$16	22	A	\$46	70	M
\$17	23	A	\$47	71	M
\$18	24	A	\$48	72	M
\$19	25		\$49	73	M
\$1A	26		\$4A	74	
\$1B	27		\$4B	75	
\$1C	28		\$4C	76	
\$1D	29		\$4D	77	
\$1E	30		\$4E	78	M
\$1F	31		\$4F	79	M
\$20	32	M	\$50	80	M A
\$21	33	M	\$51	81	M A
\$22	34	M	\$52	82	M A
\$23	35	M	\$53	83	M A
\$24	36	M	\$54	84	M A
\$25	37	M	\$55	85	M A
\$26	38	M	\$56	86	A
\$27	39	M	\$57	87	A
\$28	40	M	\$58	88	A
\$29	41	M	\$59	89	A
\$2A	42	M	\$5A	90	A
\$2B	43	M	\$5B	91	A
\$2C	44	M	\$5C	92	A
\$2D	45	M	\$5D	93	A
\$2E	46	M	\$5E	94	A
\$2F	47	M	\$5F	95	A

Table B-1—Continued. Zero Page Use

Hex	Dec	Used by	Hex	Dec	Used by
\$60	96	A I	\$90	144	A I
\$61	97	A I	\$91	145	A I
\$62	98	A I	\$92	146	A I
\$63	99	A I	\$93	147	A I
\$64	100	A I	\$94	148	A I
\$65	101	A I	\$95	149	A I
\$66	102	A I	\$96	150	A I
\$67	103	A I D	\$97	151	A I
\$68	104	A I D	\$98	152	A I
\$69	105	A I D	\$99	153	A I
\$6A	106	A I D	\$9A	154	A I
\$6B	107	A I	\$9B	155	A I
\$6C	108	A I	\$9C	156	A I
\$6D	109	A I	\$9D	157	A I
\$6E	110	A I	\$9E	158	A I
\$6F	111	A I D	\$9F	159	A I
\$70	112	A I D	\$A0	160	A I
\$71	113	A I	\$A1	161	A I
\$72	114	A I	\$A2	162	A I
\$73	115	A I	\$A3	163	A I
\$74	116	A I	\$A4	164	A I
\$75	117	A I	\$A5	165	A I
\$76	118	A I	\$A6	166	A I
\$77	119	A I	\$A7	167	A I
\$78	120	A I	\$A8	168	A I
\$79	121	A I	\$A9	169	A I
\$7A	122	A I	\$AA	170	A I
\$7B	123	A I	\$AB	171	A I
\$7C	124	A I	\$AC	172	A I
\$7D	125	A I	\$AD	173	A I
\$7E	126	A I	\$AE	174	A I
\$7F	127	A I	\$AF	175	A I D
\$80	128	A I	\$B0	176	A I D
\$81	129	A I	\$B1	177	A I
\$82	130	A I	\$B2	178	A I
\$83	131	A I	\$B3	179	A I
\$84	132	A I	\$B4	180	A I
\$85	133	A I	\$B5	181	A I
\$86	134	A I	\$B6	182	A I
\$87	135	A I	\$B7	183	A I
\$88	136	A I	\$B8	184	A I
\$89	137	A I	\$B9	185	A I
\$8A	138	A I	\$BA	186	A I
\$8B	139	A I	\$BB	187	A I
\$8C	140	A I	\$BC	188	A I
\$8D	141	A I	\$BD	189	A I
\$8E	142	A I	\$BE	190	A I
\$8F	143	A I	\$BF	191	A I

Table B-1—Continued. Zero Page Use

Hex	Dec	Used by	Hex	Dec	Used by
\$C0	192	A	\$E0	224	A
\$C1	193	A	\$E1	225	A
\$C2	194	A	\$E2	226	A
\$C3	195	A	\$E3	227	
\$C4	196	A	\$E4	228	A
\$C5	197	A	\$E5	229	A
\$C6	198	A	\$E6	230	A
\$C7	199	A	\$E7	231	A
\$C8	200	A	\$E8	232	A
\$C9	201	A	\$E9	233	A
\$CA	202	A   D	\$EA	234	A
\$CB	203	A   D	\$EB	235	
\$CC	204	A   D	\$EC	236	
\$CD	205	A   D	\$ED	237	
\$CE	206		\$EE	238	
\$CF	207		\$EF	239	
\$D0	208	A	\$F0	240	A
\$D1	209	A	\$F1	241	A
\$D2	210	A	\$F2	242	A
\$D3	211	A	\$F3	243	A
\$D4	212	A	\$F4	244	A
\$D5	213	A	\$F5	245	A
\$D6	214		\$F6	246	A
\$D7	215		\$F7	247	A
\$D8	216	A   D	\$F8	248	A
\$D9	217	A	\$F9	249	
\$DA	218	A	\$FA	250	
\$DB	219	A	\$FB	251	
\$DC	220	A	\$FC	252	
\$DD	221	A	\$FD	253	
\$DE	222	A	\$FE	254	
\$DF	223	A	\$FF	255	

## B.2 Page Three

Most of page 3 is available for small machine-language programs or any other use your program might put it to. The built-in Monitor uses the top sixteen addresses of page 3, as shown in Table B-2: the XFER routine (section 2.5.3) uses locations \$3ED and \$3EE. If you are using DOS or ProDOS, it also uses the 32 locations \$3D0 through \$3EF.

*Table B-2. Page 3 Use*

Hex	Dec	Section	Use
\$3F0	1008	2.6.4	Address of BRK request handler (normally \$59, \$FA)
\$3F1	1009		
\$3F2	1010	2.6.4 &	Reset vector
\$3F3	1011	10.1	
\$3F4	1012	2.6.4	Power-up byte (see text)
\$3F5	1013		Jump instruction to Applesoft &-command handler (initially \$4C, \$58, \$FF)
\$3F6	1014		
\$3F7	1015		
\$3F8	1016	10.6.4	Jump instruction to user CONTROL-Y command handler
\$3F9	1017		
\$3FA	1018		
\$3FB	1019		Jump instruction to NMI interrupt handler (not used by Apple IIc)
\$3FC	1020		
\$3FD	1021		
\$3FE	1022	2.6.4	Address of user IRQ interrupt handler
\$3FF	1023		

## B.3 Screen Holes

One result of the way the Apple IIc hardware maps display memory on the screen is that groups of eight memory addresses are left over in sixteen areas of the text and low-resolution display pages—eight areas in main RAM and eight in auxiliary RAM. The firmware uses for these 128 bytes are shown in Tables B-3 and B-4, with cross-references to the section numbers where they are described.

*Table B-3. Main Memory Screen Hole Allocations*

Hex	Dec	Section	Description
\$478	1144	9.1.5	Mouse port: low byte of clamping minimum
\$479	1145	7.5	Reserved for serial port 1
\$47A	1146	8.5	Reserved for serial port 2
\$47B	1147		Reserved
\$47C	1148	9.1.5	Low byte of X coordinate
\$47D	1149		Reserved for mouse port
\$47E	1150		Reserved
\$47F	1151		Reserved
\$4F8	1272	9.1.5	Mouse port: low byte of clamping maximum
\$4F9	1273	7.5,E.6.3	Reserved for serial port 1
\$4FA	1274	8.5,E.6.2	Reserved for serial port 2
\$4FB	1275		Reserved
\$4FC	1276	9.1.5	Low byte of Y coordinate
\$4FD	1277		Reserved for mouse port
\$4FE	1278		Reserved
\$4FF	1279	E.6.4	Reserved
\$578	1400	9.1.5	Mouse port: high byte of clamping minimum
\$579	1401	7.5	Port 1 printer width (1-255; 0 = unlimited)
\$57A	1402	8.5	Port 2 line length (1-255; 0 = unlimited)
\$57B	1403		Cursor horizontal position (80-column display)
\$57C	1404	9.1.5	High byte of X coordinate
\$57D	1405		Reserved for mouse port
\$57E	1406		Reserved
\$57F	1407	E.6.4	Reserved
\$5F8	1528	9.1.5	Mouse port: high byte of clamping maximum
\$5F9	1529	7.5,E.6.3	Port 1 temporary storage location
\$5FA	1530	8.5,E.6.2	Port 2 temporary storage location
\$5FB	1531		Reserved
\$5FC	1532	9.1.5	High byte of Y coordinate
\$5FD	1533		Reserved for mouse port
\$5FE	1534		Reserved
\$5FF	1535	E.6.2	Reserved



Table B-3—Continued. Main Memory Screen Hole Allocations

Hex	Dec	Section	Description
\$678	1656		Reserved
\$679	1657	7.5	Indicates when port 1 firmware is parsing a command
\$67A	1658	8.5	Indicates when port 2 firmware is parsing a command
\$67B	1659		Reserved
\$67C	1660	9.1.5	Mouse port: reserved
\$67D	1661		Reserved for mouse port
\$67E	1662		Reserved
\$67F	1663	E.6.4	Reserved
\$6F8	1784		Reserved
\$6F9	1785	7.5	Current port 1 command character
\$6FA	1786	8.5	Current port 2 command character
\$6FB	1787		Reserved
\$6FC	1788	9.1.5	Mouse port: reserved
\$6FD	1789		Reserved for mouse port
\$6FE	1790		Reserved
\$6FF	1791	E.6.2	Reserved
\$778	1912		DEVNO: \$n0 = current active port number x 16
\$779	1913	7.5	Port 1 flags for echo and auto line feed
\$77A	1914	8.5	Port 2 flags for echo and auto line feed
\$77B	1915		Reserved
\$77C	1916	9.1.5,E.6.1	Mouse port status byte
\$77D	1917		Reserved for mouse port
\$77E	1918		Reserved
\$77F	1919		Reserved
\$7F8	2040		MSLOT: owner of \$C800-\$CFFF (\$C3, video)
\$7F9	2041	7.5	Port 1 current printer column
\$7FA	2042	8.5	Port 2 current line position
\$7FB	2043		Reserved
\$7FC	2044	9.1.5	Mouse port mode byte
\$7FD	2045		Reserved for mouse port
\$7FE	2046		Reserved
\$7FF	2047		Reserved

**Table B-4. Auxiliary Memory Screen Hole Allocations**

Hex	Dec	Section	Description
\$478	1144	7.5	Initial port 1 ACIA Control Register values (\$9E)
\$479	1145	7.5	Initial port 1 ACIA Command Register values (\$0B)
\$47A	1146	7.5	Initial port 1 characteristics flags (\$40)
\$47B	1147	7.5	Initial port 1 printer width (\$50)
\$47C	1148	8.5	Initial port 2 ACIA Control Register values (\$16)
\$47D	1149	8.5	Initial port 2 ACIA Command Register values (\$0B)
\$47E	1150	8.5	Initial port 2 characteristics flags (\$01)
\$47F	1151	8.5	Initial port 2 line length (\$00)
\$4F8 through \$4FF	1272 through 1279		Reserved
\$578 through \$57F	1400 through 1407		Reserved
\$5F8 through \$5FF	1528 through 1535		Reserved
\$678 through \$67F	1656 through 1663		Reserved
\$6F8 through \$6FF	1784 through 1791		Reserved
\$778 through \$77F	1912 through 1919		Reserved
\$7F8 through \$7FF	2040 through 2047		Reserved

## B.4 The Hardware Page

Tables B-5 through B-9 list all the hardware locations available for use in the Apple IIc. These tables have a column at the left that is not present in other tables. This column, labeled RW, indicates the action to take at a particular location.

- R means read.
- RR means read twice in succession.
- R7 means read the byte and then check bit 7; in the use column, "see if..." refers to the condition represented by bit 7 = 1, unless otherwise specified. Bit 7 has a value of \$80, so if the contents of the location are greater than or equal to \$80, the bit is on.

Another way to test bit 7 (the sign bit) is with a BIT instruction, followed by BPL (bit 7 was 0) or BMI (bit 7 was 1).

- R/W means to either read or write. For writing, the value is unimportant.
- W means to write only. The value is unimportant.
- N means not to read or write, because the location is reserved.

An address of the form \$C00x means the sixteen locations from \$C000 through \$C00F. Labels, when they are shown, are simply memory aids. Some of them correspond to the labels at those addresses in the firmware, others do not. Your program will have to assign a label for it anyway.

Table B-5. Addresses \$C000 Through \$C03F

RW	Hex	Dec	Neg Dec	Label	Section	Use
R	\$C00x			KSTRB	4.1	Read keyboard data (bits 0-6) and strobe (bit 7)
W	\$C000	49152	-16384	80STORE	5.6†	Off: PAGE2 switches Page 1 and 2
W	\$C001	49153	-16383	80STORE	5.6†	On: PAGE2 switches Page 1 and 1X
W	\$C002	49154	-16382	RAMRD	2.5.2	Off: read main 48K RAM
W	\$C003	49155	-16381	RAMRD	2.5.2	On: read auxiliary 48K RAM
W	\$C004	49156	-16380	RAMWRT	2.5.2	Off: write in main 48K RAM
W	\$C005	49157	-16379	RAMWRT	2.5.2	On: write in auxiliary 48K RAM
W	\$C006	49158	-16378			Reserved
W	\$C007	49159	-16377			Reserved
W	\$C008	49160	-16376	ALTZP	2.4.2	Off: use main P0, P1, bank-switched RAM
W	\$C009	49161	-16375	ALTZP	2.4.2	On: use auxiliary P0, P1, bank-switched RAM
W	\$C00A	49162	-16374			Reserved
W	\$C00B	49163	-16373			Reserved
W	\$C00C	49164	-16372	80COL	5.6	Off: 40-column display
W	\$C00D	49165	-16371	80COL	5.6	On: 80-column display
W	\$C00E	49166	-16270	ALTCHAR	5.6	Off: display primary character set
W	\$C00F	49167	-16369	ALTCHAR	5.6	On: display alternate character set
W	\$C01x				4.1	Clear keyboard strobe (\$C00x bit 7)
R7	\$C010	49168	-16368	AKD	4.1	See if any key now down; clear strobe
R7	\$C011	49169	-16367	RDBNK2	2.4.2	See if using SD000 bank 2 (or 1)
R7	\$C012	49170	-16366	RDLGRAM	2.4.2	See if reading RAM (or ROM).
R7	\$C013	49171	-16365	RDRAMRD	2.5.2	See if reading auxiliary 48K RAM (or main)
R7	\$C014	49172	-16364	RDRAMWRT	2.5.2	See if writing auxiliary 48K RAM (or main)
R	\$C015	49173	-16363	RSTXINT	9.1.3	Reset mouse X0 interrupt.
R7	\$C016	49174	-16362	RDALTZP	2.4.2	See if auxiliary P0, P1 and bank-switched RAM
R	\$C017	49175	-16361	RSTYINT	9.1.3	Reset mouse Y interrupt
R7	\$C018	49176	-16360	RD80STORE	5.6†	See if 80STORE on (or off)
R7	\$C019	49177	-16359	RSTVBL	9.1.3	See if VBLINT off (1); reset it
R7	\$C01A	49178	-16358	RDTEXT	5.6	See if text (or graphics)
R7	\$C01B	49179	-16357	RD MIX	5.6	See if mixed mode switch on
R7	\$C01C	49180	-16356	RDPAGE2	5.6†	See if page 2/1X selected (or 1)
R7	\$C01D	49181	-16355	RDHRES	5.6†	See if high-resolution switch on
R7	\$C01E	49182	-16354	RDALTCHAR	5.6	See if alternate character set (or primary)
R7	\$C01F	49183	-16353	RD80COL	5.6	See if 80-column hardware on
N	\$C020	49184	-16352			Reserved (read and write)
N	through \$C02F	49199	-16337			Reserved
W	\$C030	49200	-16336		4.2.1	Toggle speaker
R	\$C030	49200	-16336			
N	\$C031	49201	-16335			Reserved (read and write)
N	through \$C03F	49215	-16321			Reserved

† Also section 2.5.4

Table B-6. Addresses \$C040 Through \$C05F

RW	Hex	Dec	Neg Dec	Label	Section	Use
R7	\$C040	49216	-16320	RDXYSK	9.1.3	See if X0/Y0 mask set
R7	\$C041	49217	-16319	RDVBLMSK	9.1.3	See if VBL mask set
R7	\$C042	49218	-16318	RDX0EDGE	9.1.3	See if interrupt on falling X0 edge
R7	\$C043	49219	-16317	RDY0EDGE	9.1.3	See if interrupt on falling Y0 edge
N	\$C044	49220	-16316			Reserved
N	\$C045	49221	-16315			Reserved
N	\$C046	49222	-16314			Reserved
N	\$C047	49223	-16313			Reserved
R	\$C048	49224	-16312	RSTXY	9.1.3	Reset X0/Y0 interrupt flags
N	\$C049	49225	-16311			Reserved
N	\$C04A	49226	-16310			Reserved
N	\$C04B	49227	-16309			Reserved
N	\$C04C	49228	-16308			Reserved
N	\$C04D	49229	-16307			Reserved
N	\$C04E	49230	-16306			Reserved
N	\$C04F	49231	-16305			Reserved
R/W	\$C050	49232	-16304	TEXT	5.6	Off: graphics display
R/W	\$C051	49233	-16303	TEXT	5.6	On: text display
R/W	\$C052	49234	-16302	MIXED	5.6	Off: text or graphics only
R/W	\$C053	49235	-16301	MIXED	5.6	On: combination text and graphics
R/W	\$C054	49236	-16300	PAGE2	5.6†	Off: use page 1
R/W	\$C055	49237	-16299	PAGE2	5.6†	On: display page 2 (80STORE off), store to page 1X (80STORE on)
R/W	\$C056	49238	-16298	HIRES	5.6†	Off: low-resolution
R/W	\$C057	49239	-16297	HIRES	5.6†	On: high-resolution; double if 80COL and DHIRES on
N	\$C058	49240	-16296			Reserved if IOUDIS on (\$C07E bit 7=1)
R/W				DISXY	9.1.3	Disable (mask) mouse X0/Y0 interrupts
N	\$C059	49241	-16295			Reserved if IOUDIS on
R/W				ENBXY	9.1.3	Enable (allow) mouse X0/Y0 interrupts
N	\$C05A	49242	-16294			Reserved if IOUDIS on
R/W				DISVBL	9.1.3	Disable (mask) VBL interrupts
N	\$C05B	49243	-16293			Reserved if IOUDIS on
R/W				ENVBL	9.1.3	Enable (allow) VBL interrupts
N	\$C05C	49244	-16292			Reserved if IOUDIS on
R/W				X0EDGE	9.1.3	Interrupt on rising edge of X0
N	\$C05D	49245	-16291			Reserved if IOUDIS on
R/W				X0EDGE	9.1.3	Interrupt on falling edge of X0
R/W	\$C05E	49246	-16290	DHIRES	5.6	If IOUDIS on: set double-high-resolution
R/W				Y0EDGE	9.1.3	If IOUDIS off: interrupt on rising Y0
R/W	\$C05F	49247	-16289	DHIRES	5.6	If IOUDIS on: clear double-high-resolution
R/W				Y0EDGE	9.1.3	If IOUDIS off: interrupt on falling Y0

† Also section 2.5.4.

**Table B-7. Addresses \$C060 Through \$C07F**

RW	Hex	Dec	Neg Dec	Label	Section	Use
W	\$C06x					Reserved (write)
R7	\$C060	49248	-16288	RD80SW	4.1	See if 80/40 switch down (= 40)
R7	\$C061	49249	-16287	RDBTN0	9.1.3†	See if switch 0 or (⬇) pressed
R7	\$C062	49250	-16286	RDBTN1	9.2†	See if switch 1 or (⬆) pressed
R7	\$C063	49251	-16285	RD63	9.1,9.2	See if mouse button not pressed
R7	\$C064	49252	-16284	PDL0	9.2	See if hand control button 0 pressed
R7	\$C065	49253	-16283	PDL1	9.2	See if hand control button 1 pressed
R7	\$C066	49254	-16282	MOUX1	9.1.3	See if mouse X1 (direction) is high
R7	\$C067	49255	-16281	MOUY1	9.1.3	See if mouse Y1 (direction) is high
N	\$C068	49256	-16280			Reserved (write and read)
	through					
N	\$C06F	49263	-16273			
R/W	\$C07x					Trigger paddle timer; reset VBLINT; however, some \$C07x are reserved
R/W	\$C070	49264	-16272	PTRIG	9.2	Designated trigger or reset location
N	\$C071	49265	-16271			Reserved
	through					
N	\$C07D	49277	-16259			
R7	\$C07E	49278	-16258	RDIODIS		See if IODIS on; trigger paddle timer; reset VBLINT
W				IODIS	5.6,9.1.3	On: enable access to DHIRES switch; disable \$C058-\$C05F IOU access
R7	\$C07F	49279	-16257	RDDHIRES	5.6,9.1.3	See if DHIRES on
W				IODIS	5.6	Off: disable access to DHIRES switch; enable \$C058-\$C05F IOU access

† Also section 4.1.



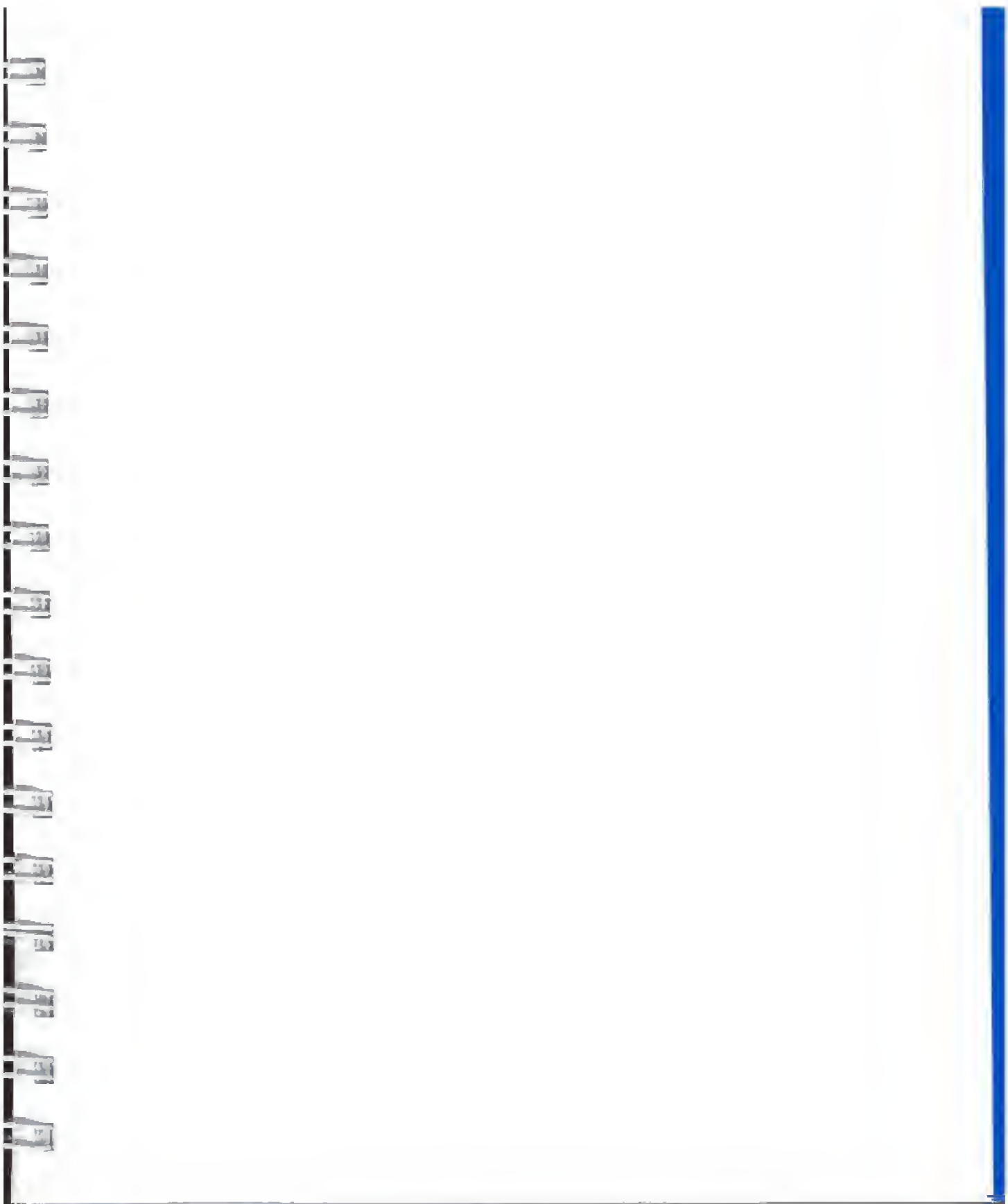
**Table B-8. Addresses \$C080 Through \$C0AF**

RW	Hex	Dec	Neg Dec	Label	Section	Use
R	\$C080	49280	-16256		2.4.2	Read RAM; no write; use \$D000 bank 2
RR	\$C081	49281	-16255		2.4.2	Read ROM, write RAM; use \$D000 bank 2
R	\$C082	49282	-16254		2.4.2	Read ROM, no write; use \$D000 bank 2
RR	\$C083	49283	-16253		2.4.2	Read and write RAM; use \$D000 bank 2
N	\$C084	49284	-16252			Reserved
N	\$C085	49285	-16251			Reserved
N	\$C086	49286	-16250			Reserved
N	\$C087	49287	-16249			Reserved
R	\$C088	49288	-16248		2.4.2	Read RAM; no write; use \$D000 bank 1
RR	\$C089	49289	-16247		2.4.2	Read ROM, write RAM; use \$D000 bank 1
R	\$C08A	49290	-16246		2.4.2	Read ROM, no write; use \$D000 bank 1
RR	\$C08B	49291	-16245		2.4.2	Read and write RAM; use \$D000 bank 1
N	\$C08C	49292	-16244			Reserved
N	\$C08D	49293	-16243			Reserved
N	\$C08E	49294	-16242			Reserved
N	\$C08F	49295	-16241			Reserved
N	\$C090	49296	-16240			Reserved
N	\$C097	49303	-16233			Reserved
R/W	\$C098	49304	-16232		7.3, 11.11	Port 1 ACIA Transmit/receive register
R/W	\$C099	49305	-16231		7.3, 11.11	Port 1 ACIA Status register
R/W	\$C09A	49306	-16230		7.3, 11.11, Appendix E	Port 1 ACIA Command register
R/W	\$C09B	49307	-16229		7.3, 11.11	Port 1 ACIA Control register
N	\$C09C	49308	-16228			Reserved
N	\$C09F	49311	-16225			Reserved
N	\$C0A0	49312	-16224			Reserved
N	\$C0A7	49319	-16217			Reserved
R/W	\$C0A8	49320	-16216		8.3, 11.11	Port 2 ACIA Transmit/receive register
R/W	\$C0A9	49321	-16215		8.3, 11.11	Port 2 ACIA Status register
R/W	\$C0AA	49322	-16214		8.3, 11.11, Appendix E	Port 2 ACIA Command register
R/W	\$C0AB	49323	-16213		8.3, 11.11	Port 2 ACIA Control register
N	\$C0AC	49324	-16212			Reserved
N	\$C0AF	49327	-16209			Reserved



**Table B-9. Addresses \$C0B0 Through \$C0FF**

RW	Hex	Dec	Neg Dec	Label	Section	Use
N	\$C0B0	49328	-16208			Reserved
N	\$C0BF	49343	-16193			
N	\$C0C0	49344	-16192			Reserved
N	\$C0CF	49359	-16177			
N	\$C0D0	49360	-16176			Reserved
N	\$C0DF	49375	-16161			
N	\$C0E0	49376	-16160			Reserved
N	\$C0EF	49391	-16145			
N	\$C0F0	49392	-16144			Reserved
N	\$C0FF	49407	-16129			



# *Important Firmware Locations*

This appendix lists all significant firmware addresses: entry points, locations containing the addresses of entry points, and locations where machine and device identification bytes reside.



#### Warning

*The Monitor firmware entry points are the only published entry points in the sense that they are the only ones that will remain in the same locations in future Apple II series computers.*

*The firmware protocol identification bytes and offsets will work with other Apple II series computers only if used as directed (section 3.4.2).*

## C.1 The Tables

Appendix H contains tables and examples of the derivation of each form of address from either of the other forms.

This appendix supplements the chapter text by specifying three forms of each address: hexadecimal, decimal, and complementary (negative) decimal.

In these tables, some of the addresses are followed by a label of the location. These labels are listed only to assist you in finding the named location in the firmware listings, or in remembering the function found at the address. The Apple IIc contains no global label table: your program must assign its own labels to the addresses as required.

There are several types of information at these firmware addresses: actual entry points (labeled *entry*), the low-order byte of an entry point (labeled *offset*), a device or machine

identification byte (labeled *ident*), indicators (labeled *indic*) specifying whether there are optional routines, vector addresses (labeled *vector*), and an RTS instruction location.

The column labeled *Section* contains the number of the section that describes the item. If there is no description except in a table in this appendix, a section number is not listed.

Each input/output port has an associated protocol table, as shown in Tables C-1 through C-4. Many of the bytes (labeled *offset*) in the protocol tables are the low-order bytes of addresses of I/O routines for the ports; the high-order byte of these addresses must be \$Cn (where *n* is the port number). This structure is explained in Chapter 3. Although your program must perform some extra processing to use these tables, the benefit is simplified compatible port and slot I/O for all Apple II series machines.

## C.2 Port Addresses

Table C-1. Serial Port 1 Addresses

Hex	Dec	Neg Dec	Label	Type	Section	Description
\$C100	49408	-16128		entry	3.1.1	Main port 1 entry point
\$C105	49413	-16123		ident	3.4.2	ID byte (\$38)
\$C107	49415	-16121		ident	3.4.2	ID byte (\$18)
\$C10B	49419	-16117		ident	3.4.2	Firmware card signature (\$01)
\$C10C	49420	-16116		ident	3.4.2	Super Serial Card ID (\$31)
\$C10D	49421	-16115		offset	7.4	Low-order PINIT address
\$C10E	49422	-16114		offset	7.4	Low-order PREAD address
\$C10F	49423	-16113		offset	7.4	Low-order PWRITE address
\$C110	49424	-16112		offset	7.4	Low-order PSTATUS address
\$C111	49425	-16111		indic	3.4.2	Non-zero: no optional routines

Table C-2. Serial Port 2 Addresses

Hex	Dec	Neg Dec	Label	Type	Section	Description
\$C200	49664	-15872		entry	3.1.1	Main port 2 entry point
\$C205	49669	-15867		ident	3.4.2	ID byte (\$38)
\$C207	49671	-15865		ident	3.4.2	ID byte (\$18)
\$C20B	49675	-15861		ident	3.4.2	Firmware card ID (\$01)
\$C20C	49676	-15860		ident	3.4.2	Super Serial Card ID (\$31)
\$C20D	49677	-15859		offset	8.4	Low-order PINIT address
\$C20E	49678	-15858		offset	8.4	Low-order PREAD address
\$C20F	49679	-15857		offset	8.4	Low-order PWRITE address
\$C210	49680	-15856		offset	8.4	Low-order PSTATUS address
\$C211	49681	-15855		indic	3.4.2	Non-zero: no optional routines

**Table C-3. Video Firmware Addresses**

Hex	Dec	Neg Dec	Label	Type	Section	Description
\$C300	49920	-15616		entry	3.1.1	Main video entry point (output only)
\$C305	49925	-15611	C3KEYIN	ident	3.4.2	ID byte (\$38)
\$C307	49927	-15609	C3COUT1	ident	3.4.2	ID byte (\$18)
\$C30B	49931	-15605		ident	3.4.2	Firmware card signature (\$01)
\$C30C	49932	-15604		ident	3.4.2	80-column card ID (\$88)
\$C30D	49933	-15603		offset	5.9	Low-order PINIT address
\$C30E	49934	-15602		offset	5.9	Low-order PREAD address
\$C30F	49935	-15601		offset	5.9	Low-order PWRITE address
\$C310	49936	-15600		offset	5.9	Low-order PSTATUS address
\$C311	49937	-15599	MOVEAUX	entry	2.5.3	Routine for main/auxiliary control swapping (Also called AUXMOVE)

**Table C-4. Mouse Port Addresses**

Hex	Dec	Neg Dec	Label	Type	Section	Description
\$C400	50176	-15360		entry		Main mouse entry point
\$C405	50181	-15355		ident	3.4.2	ID byte (\$38)
\$C407	50183	-15353		ident	3.4.2	ID byte (\$18)
\$C40B	50187	-15349		ident	3.4.2	Firmware card signature (\$01)
\$C40C	50188	-15348		type	3.4.2	X-Y pointing device ID (\$20)
\$C40D	50189	-15347		offset	9.1.4	Low-order PINIT address
\$C40E	50190	-15346		offset	9.1.4	Low-order PREAD address
\$C40F	50191	-15345		offset	9.1.4	Low-order PWRITE address
\$C410	50192	-15344		offset	9.1.4	Low-order PSTATUS address
\$C411	50193	-15343		indic	3.4.2	Optional routines follow (\$00)
\$C412	50194	-15342	SETMOUSE	offset	9.1.4	Low-order SETMOUSE address
\$C413	50195	-15341	SERVEMOUSE	offset	9.1.4	Low-order SERVEMOUSE address
\$C414	50196	-15340	READMOUSE	offset	9.1.4	Low-order READMOUSE address
\$C415	50197	-15339	CLEARMOUSE	offset	9.1.4	Low-order CLEARMOUSE address
\$C416	50198	-15338	POSMOUSE	offset	9.1.4	Low-order POSMOUSE address
\$C417	50199	-15337	CLAMPMOUSE	offset	9.1.4	Low-order CLAMPMOUSE address
\$C418	50200	-15336	HOMEMOUSE	offset	9.1.4	Low-order HOMEMOUSE address
\$C419	50201	-15335	INITMOUSE	offset	9.1.4	Low-order INITMOUSE address

---

### **C.3 Other Video and I/O Firmware Addresses**

Miscellaneous firmware addresses are listed in Table C-5.

*Table C-5. Apple IIc Enhanced Video and Miscellaneous Firmware*

Hex	Dec	Neg Dec	Label	Type	Section	Description
\$C800	50688	-14848		entry	6.1	Disk drive firmware entry point
\$C700	50944	-14592		entry	6.2	External disk startup routine
\$C803	51203	-14333	NEWIRQ	entry	E.1	IRQ handling routine

---

### **C.4 Applesoft BASIC Interpreter Addresses**

The addresses of Applesoft BASIC entry points are listed in the *Applesoft BASIC Programmer's Reference Manual*. The Applesoft interpreter occupies ROM addresses from \$D000 through \$F7FF.

---

### **C.5 Monitor Addresses**

Table C-6 lists the Monitor entry points, machine identifier bytes, interrupt vectors, and the address of a known RTS instruction.



Table C-6. Apple IIc Monitor Entry Points and Vectors

Hex	Dec	Neg Dec	Label	Type	Section	Description
\$F800	63488	-2048	PLOT	entry	5.8	Plots a low-resolution block
\$F819	63513	-2023	HLINE	entry	5.8	Draws low-resolution horizontal line
\$F828	63528	-2008	VLINE	entry	5.8	Draws low-resolution vertical line
\$F832	63538	-1998	CLRSCR	entry	5.8	Clears low-resolution screen
\$F836	63542	-1994	CLRTOP	entry	5.8	Clears top 40 low-resolution lines
\$F864	63588	-1948	SETCOL	entry	5.8	Sets low-resolution color (Table 5-4)
\$F871	63601	-1935	SCRN	entry	5.8	Reads color of low-resolution block
\$F941	63809	-1727	PRNTAX	entry	5.8	Displays (A) and (X) in hex
\$F94A	63818	-1718	PRBL2	entry	5.8	Sends (X) blanks to output
\$FA47	63845	-1691	NEWBRK	entry	E.2	Apple IIc break handler
\$FA62	64098	-1438	RESET	entry	2.6	Hardware reset routine
\$FB1E	64286	-1250	PREAD	entry	9.2	Reads hand control position
\$FB6F	64367	-1169	SETPWRC	entry	2.6.4	Routine to create power-up byte
\$FB63	64435	-1101		ident	F.1.2	Machine identification byte
\$FBC0	64448	-1088		ident	F.1.2	Machine identification byte
\$FBDD	64477	-1059	BELL1	entry	4.2.2	Sends 1 kHz beep to speaker
\$FC42	64578	-958	CLREOP	entry	5.8	Clears from cursor to bottom
\$FC58	64600	-936	HOME	entry	5.8	Clears; cursor to upper left
\$FC9C	64668	-868	CLREOL	entry	5.8	Clears from cursor to end of line
\$FC9E	64670	-866	CLEOLZ	entry	5.8	Clears from BASL to end of line
\$FCA8	64680	-856	WAIT	entry		Delays for time specified by (A)
\$FD0C	64780	-756	RDKEY	entry	3.2.1	Displays cursor, jumps to (KSW)
\$FD1B	64795	-741	KEYIN	entry	3.2.2	Waits for keypress, reads key
\$FD35	64821	-715	RDCHAR	entry	4.1.2	Gets input, interprets ESC codes
\$FD67	64871	-665	GETLNZ	entry	4.1.2	Sends CR to output, goes to GETLN
\$FD6A	64874	-662	GETLN	entry	3.2.3	Displays prompt, gets input line
\$FD6F	64879	-657	GETLN1	entry	4.1.2	No prompt; gets input line
\$FD8B	64907	-629	CROUT1	entry	5.8	Clears to end of line, calls CROUT
\$FD8E	64910	-626	CROUT	entry	5.8	Sends CR to output
\$FDDA	64986	-550	PRBYTE	entry	5.8	Sends (A) to output
\$FDE3	64995	-541	PRHEX	entry	5.8	Displays low nibble of (A) in hex
\$FDED	65005	-531	COUT	entry	3.3.1	Jumps to (CSW)
\$FDF0	65008	-528	COUT1	entry	3.3.2	Displays (A), advances cursor
\$FE2C	65068	-468	MOVE	entry		Copies (memory) elsewhere
\$FE36	65078	-458	VERIFY	entry		Compares two blocks of memory
\$FF2D	65325	-211	PRERR	entry	5.8	Sends ERR to output; beeps
\$FF3A	65338	-198	BELL	entry	4.2.2	Sends CONTROL-G to output
\$FF3F	65343	-193	IOREST	entry		Loads (\$45-\$49) into registers
\$FF4A	65354	-182	IOSAVE	entry		Stores (A,X,Y,P,S) at \$45-\$49
\$FF58	65368	-168	IORTS	RTS		Location of known RTS instruction
\$FF69	65385	-151	(Monitor)	entry	10.1	Standard Monitor entry point
\$FFFA	65530	-6		vector		Low-order NMI vector (unused)
\$FFFB	65531	-5		vector		High-order NMI vector (unused)
\$FFFC	65532	-4		vector		Low-order RESET vector (\$62)
\$FFFD	65533	-3		vector		High-order RESET vector (\$FA)
\$FFFE	65534	-2	IRQVECT	vector		Low-order IRQ vector (\$03)
\$FFFF	65535	-1		vector		High-order IRQ vector (\$CB)

# *Operating Systems and Languages*



This appendix is an overview of the characteristics of operating systems and languages when run on the Apple IIc. It is not intended to be a full account. For more information, refer to the manuals that are provided with each product.

---

## **D.1 Operating Systems**

This section discusses the operating systems that the Apple IIc does and does not work with.

---

### **D.1.1 ProDOS**

ProDOS is the preferred disk operating system for the Apple IIc. It supports startup from the external disk drive, interrupts, and all other hardware and firmware features of the Apple IIc.

---

### **D.1.2 DOS**

The Apple IIc works with DOS 3.3. Its disk drive support hardware and firmware can also access DOS 3.2 disks by using the *BASIC* disk. However, neither version of DOS takes full advantage of the features of the Apple IIc. DOS support is provided only for the sake of Apple II series compatibility.

---

### ***D.1.3 Pascal Operating System***

Version 1.2 of the Pascal Operating System uses the 80/40 switch and the interrupt features of the Apple IIc, while remaining compatible with the other Apple II series computers.

While the Apple IIc works with Pascal 1.1, this version of the Pascal Operating System does not use the 80/40 switch or handle interrupts.

The Apple IIc does not work with Pascal 1.0, because the input/output firmware entry points are rigidly defined (rather than being accessed via a table), and the firmware does not support these entry points.

---

### ***D.1.4 CP/M***

CP/M, and any other operating system that requires an interface card, will not work on the Apple IIc.

---

## ***D.2 Languages***

For further information about these languages, refer to the manuals that came with them.

This section discusses special techniques to use, and characteristics to be aware of, when using Apple programming languages with the Apple IIc. It is also a guide to using this reference manual with these languages.

---

### ***D.2.1 Applesoft BASIC***

The focus of the chapters in this manual is assembly language, and so most addresses and values are given in hexadecimal notation.

Use the appendixes to make or find decimal conversions. Appendix H has tables and examples to help you convert numbers between hexadecimal, decimal, and negative (complementary) decimal. All the addresses listed in Appendixes B and C—screen holes, hardware addresses, firmware entry points, and so on—are given in all three numeric forms.

Use a PEEK in BASIC (instead of LDA in assembly language) to read a location, and a POKE (instead of STA) to write to a location. If you read a hardware address from a BASIC program, you get a value between 0 and 255. Bit 7 has a value of 128, so if a soft switch is on, its value will be equal to or greater than 128; if the switch is off, the value will be less than 128.

---

### ***D.2.2 Integer BASIC***

Unless you load a version of DOS into your Apple IIc, you will not have Integer BASIC available inside the machine. ProDOS does not support Integer BASIC.

---

### ***D.2.3 Pascal Language***

The Pascal language works on the Apple IIc under versions 1.1 and 1.2 of the Pascal Operating System. However, for best performance, use Pascal version 1.2.

---

### ***D.2.4 FORTRAN***

FORTRAN works under version 1.1 of the Pascal Operating System which, as explained in section D.1.3, does not detect or use certain Apple IIc features, such as the 80/40 switch or auxiliary memory. Therefore, FORTRAN does not take advantage of these features either.

# *Interrupts*

11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100



This appendix presents a unified account of the sources of interrupts on the Apple IIc, how the firmware handles the interrupts, and how to use interrupt-driven features directly in those rare cases when the firmware cannot meet your needs.

**Warning**

*If you use interrupt hardware directly, rather than using the built-in interrupt-handling firmware, compatibility with possible future Apple II series computers or revisions cannot be guaranteed.*

## **E.1 Introduction**

This section orients you to interrupts and their effects on the Apple IIc hardware.

### **E.1.1 What Is an Interrupt?**

On a computer, an interrupt is a signal that abruptly causes the computer to stop what it is currently doing and immediately attend to an important time-dependent task. For example, the Apple IIc mouse sends an interrupt to the computer every time it moves. This is necessary because unless the mouse is read shortly after it moves, the signal indicating its direction is lost.

When an interrupt occurs, control passes to an interrupt handler, which must record the exact state of the computer at the moment of the interrupt, determine the source of the interrupt, and take appropriate action. It is important that the

computer preserve a *snapshot* of its state when interrupted, so that when it continues later with what it had been doing, those conditions can be restored.

---

### ***E.1.2 Interrupts on Apple II Computers***

Interrupts have not always been fully supported on the Apple II. All versions of Apple's DOS, as well as the Monitor program, rely on the integrity of location \$45, which the built-in interrupt handler has always destroyed by saving the accumulator in it. Most versions of Pascal simply do not work with interrupts enabled.

The Apple IIc built-in interrupt handler now saves the accumulator on the stack instead of in location \$45. Thus both DOS and the Monitor work with interrupts on the Apple IIc.

If, however, you want software that uses interrupts to work on the Apple IIe and the Apple II Plus, you must use either ProDOS, Apple's new enhanced disk operating system, or Pascal 1.2. Both operating systems have full interrupt support built in.

Interrupts are effective only if they are enabled most of the time. Interrupts that occur while interrupts are disabled cannot be detected. Due to the critical timing of disk read and write operations, Pascal, DOS, and ProDOS turn off interrupts while accessing the disk. Thus it is important to remember that while a disk drive is being accessed, all sources of interrupts discussed below are turned off.

On the Apple IIe only, interrupts are periodically turned off while 80-column screen operations are being performed. This is most noticeable while the screen is scrolling. Also, most peripheral cards used in the Apple IIe disable interrupts while reading and writing.

---

### ***E.1.3 Interrupt Handling on the 65C02***

From the point of view of the 65C02, there are three possible causes of interrupts.

1. If 65C02 interrupts are not masked (that is, the CLI instruction has been used), the IRQ line on the microprocessor can be pulled low. This is the standard technique by which a device indicates that it needs immediate attention.
2. The processor executes a break (BRK, opcode \$00) instruction.
3. A non-maskable interrupt (NMI) occurs. Because the NMI line in the Apple IIc's 65C02 is not used, this never happens.

Options 1 and 2 cause the 65C02 to save the current program counter and status byte on the stack and then jump to the routine whose address is stored in \$FFFE and \$FFFF. The sequence performed by the 65C02 is:

1. If IRQ, finish executing the current instruction. (If BRK, current instruction is already finished.)
2. Push high byte of program counter onto stack.
3. Push low byte of program counter onto stack.
4. Push program status byte onto stack.
5. Jump to address stored in \$FFFE, \$FFFF, that is, JMP (\$FFFE).

The different sources of interrupt signals are discussed below.

---

### ***E.1.4 The Interrupt Vector at \$FFFE***

In the Apple IIc computer there are three separate regions of memory that contain address \$FFFE: the built-in ROM, the bank-switched memory in main RAM, and the bank-switched memory in auxiliary RAM. The vector at \$FFFE in the ROM points to Apple IIc's built-in interrupt handling routine. Due to the complexity of interrupts in the Apple IIc, it is recommended that you use it rather than writing your own interrupt handling routine.

When you initialize the mouse or serial communication firmware, copies of the ROM's interrupt vector are placed in the interrupt vector addresses in both main and auxiliary bank-switched

memory. If you plan to use interrupts and the bank-switched memory without the mouse or communication firmware, you must copy the ROM's interrupt vector yourself.

## **E.2 The Built-in Interrupt Handler**

The built-in interrupt handler is responsible for determining whether a BRK or an IRQ interrupt occurred. If it was an IRQ interrupt, it decides whether the interrupt should be handled internally, handled by the user, or simply ignored.

The built-in interrupt handling routine records the current memory configuration, then sets up its own standard memory configuration so that a user's interrupt handler knows the precise memory configuration when it is called.

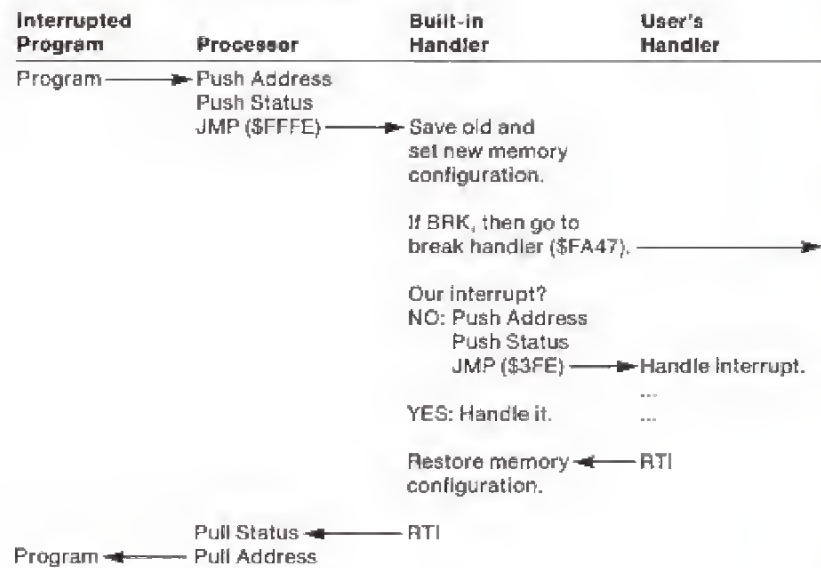
Next the handler checks to see if the interrupt was caused by a break instruction, and if it was, handles it as described in section E.4.

If the interrupt was not caused by a BRK, the handler checks for interrupts that it knows how to handle (for example, a properly initialized mouse) and handles them.

Depending on the state of the system, it either ignores other interrupts, or passes them to a user's interrupt handling routine whose address is stored at \$3FE and \$3FF of main memory.

After handling an interrupt itself, or after the user's handler returns (with an RTI), the built-in interrupt handler restores the memory configuration, and then does an RTI to restore processing to where it was when the interrupt occurred. Figure E-1 illustrates this whole process. Each of the steps is explained in detail in the sections that follow.

Figure E-1. Interrupt-Handling Sequence





---

### ***E.2.1 Saving the Memory Configuration***

The built-in interrupt handler saves the state of the system, and sets it to a known state according to these rules:

- If 80STORE and PAGE2 are on, then it switches in Text Page 1 (PAGE2 off) so that main screen holes are accessible.
- It switches in main memory for reading (RAMRD off).
- It switches in main memory for writing (RAMWRT off).
- It switches in ROM addresses \$D000-\$FFFF for reading (RDLGRAM off).
- It switches in main stack and zero page (ALTZP off).
- It preserves the auxiliary stack pointer, and restores the main stack pointer (see section E.2.2).

**Note:** Because main memory is switched in, all memory addresses used later in this appendix are in main memory unless otherwise specified.

---

### ***E.2.2 Managing Main and Auxiliary Stacks***

Because the Apple IIc has two stack pages, the firmware has established a convention that allows the system to be run with two separate stack pointers. Two bytes in the auxiliary stack page are to be used as storage for inactive stack pointers: \$100 for the main stack pointer when the auxiliary stack is active, and \$101 for the auxiliary stack pointer when the main stack is active.

When a program that uses interrupts switches in the auxiliary stack for the first time, it should place the value of the main stack pointer at auxiliary stack address \$100, and initialize the auxiliary stack pointer to \$FF (the top of the stack). When it subsequently switches from one stack to the other, it should save the current stack pointer before loading the pointer for the other stack.

When an interrupt occurs while the auxiliary stack is switched in, the current stack pointer is stored at \$101, and the main stack pointer is retrieved from \$100. Then the main stack is switched in for use. After the interrupt has been handled, the stack pointer is restored to its original value.



### **E.3 User's Interrupt Handler at \$3FE**

Screen hole locations can be set up to indicate that the user's interrupt handler should be called when certain interrupts occur. To use such a routine, place its address at \$3FE and \$3FF in main memory, low byte first.

The user's interrupt handler should do the following:

- Verify that the interrupt came from the expected source. The following sections describe how this should be done for each interrupt source.
- Handle the interrupt as desired.
- Clear the interrupt, if necessary. The following sections describe how to clear the interrupts.
- Return using an RTI instruction.

If your interrupt handler needs to know the memory configuration at the time of the interrupt, it can check the encoded byte stored four bytes down on the stack. This byte is explained in section E.4.

In general there is no guaranteed response time for interrupts. This is because the system may be doing a disk operation, which could last for several seconds.

Once the built-in interrupt handler has been called, it takes about 250 to 300 microseconds for it to call your interrupt handling routine. After your routine returns, it takes 40 to 140 microseconds to restore memory and return to the interrupted program.

If memory is in the standard state when the interrupt occurs, the total overhead for interrupt processing is about 150 microseconds less than if memory is in the worst possible state (80STORE and PAGE2 on, auxiliary memory switched in for reading and writing, auxiliary bank-switched memory page 2 switched in for reading and writing).

## **E.4 Handling Break Instructions**

After the interrupt handler has set the memory configuration, it checks to see if the interrupt was caused by a BRK (opcode \$00) instruction. (If it was, bit 4 of the processor status byte is a one). If so, it jumps to a break handling routine, which saves the state of the computer at the time of the break as follows.

Information	Location
Program counter (low byte)	\$3A
Program counter (high byte)	\$3B
Encoded memory state	\$44
Accumulator	\$45
X register	\$46
Y register	\$47
Status register	\$48

Finally the break routine jumps to the routine whose address is stored at \$3F0 and \$3F1.

The encoded memory state in location \$44 can be interpreted as follows:

Bit 7	=	0	
Bit 6	=	1	if 80STORE and PAGE2 both on
Bit 5	=	1	if auxiliary RAM switched in for reading
Bit 4	=	1	if auxiliary RAM switched in for writing
Bit 3	=	1	if bank-switched RAM being read
Bit 2	=	1	if bank-switched \$D000 page 1 switched in
Bit 1	=	1	if bank-switched \$D000 page 2 switched in
Bit 0	=	0	

## ***E.5 Sources of Interrupts***

The Apple IIc can receive interrupts from many different sources. Each source is enabled and used slightly differently from the others. There are two basic classes of interrupt sources: those associated with use of the mouse, and those associated with the two 6551 ACIA circuits (the chips that control serial communication).

The interrupts that are associated with the mouse are these:

- An interrupt can be generated when the mouse is moved in the horizontal (X) direction.
- An interrupt can be generated when the mouse is moved in the vertical (Y) direction.
- An interrupt can be generated every 1/60 second. This is called the **vertical blanking** (VBL) interrupt, and is synchronized with a signal used for the video display.
- Using the firmware, interrupts can be generated when the mouse button is pressed.

The interrupts that are associated with the ACIAs are these:

- An interrupt can be generated when a key is pressed. The firmware can use this interrupt to buffer keystrokes, or it can pass the interrupt on to the user.
- An interrupt can be generated by a device attached to the external disk drive port. The firmware can pass this interrupt on to the user.
- An interrupt can be generated when either ACIA has received a byte of data from its port. The firmware can use this interrupt to buffer data or it can pass the interrupt on to the user.
- An interrupt can be generated when pin 5 of either serial port changes state (device ready/not ready to accept data). When the serial firmware is active, this interrupt is absorbed; however, the serial firmware uses the signal to decide whether or not to transmit the next byte of data.
- An interrupt can be generated when either ACIA is ready to accept another character to be transmitted. When the serial firmware is active, this interrupt is absorbed; however, the serial firmware uses the signal to decide whether or not to transmit the next byte of data.
- An interrupt is generated when the keyboard strobe is cleared. The firmware absorbs this interrupt.

---

## **E.6 Firmware Handling of Interrupts**

The following sections discuss the various sources of interrupts and how they should be used in conjunction with the built-in interrupt handler.

---

### **E.6.1 Firmware for Mouse and VBL**

As described in Chapter 9, the mouse can be initialized (by the SETMOUSE call) to nine different modes that enable one or more sources of interrupts. In transparent mode, the interrupts are entirely handled by the built-in interrupt handler; the other modes require a user-installed interrupt handler.

When the mouse is initialized, the interrupt vector is copied to addresses \$FFFE and \$FFFF in main and auxiliary bank-switched RAM. This permits mouse interrupts with any memory configuration.

When the mouse is active, possible sources of interrupts are:

- Mouse movement in the X direction
- Mouse movement in the Y direction
- Change of state of the button
- Rising edge of the vertical blanking signal.

When an interrupt occurs, the built-in interrupt handler determines whether that particular interrupt source was enabled (by the SETMOUSE call). If so, the user's interrupt handler, whose address is stored at \$3FE, is called.

The user's interrupt handler should first call SERVEMOUSE to determine the source of the interrupt. This call updates the mouse status byte at \$77C and returns with the carry bit clear if mouse movement, button, or vertical blanking was the source of the interrupt.

The values of this mouse status byte at \$77C are as follows:

Bit	1 means that
3	Interrupt was from vertical blanking
2	Interrupt was from button
1	Interrupt was from mouse movement

If the interrupt was due to mouse movement or button, the user's interrupt handler should then do a call to READMOUSE. This causes the mouse coordinates and status to be updated as follows:

\$47C	Low byte of X coordinate
\$4FC	Low byte of Y coordinate
\$57C	High byte of X coordinate
\$5FC	High byte of Y coordinate



\$77C Button and movement status

Bit	Means
7	0 = button up; 1 = button down
6	0 = button up on last READMOUSE 1 = button down on last READMOUSE
5	0 = no movement since last READMOUSE 1 = movement since last READMOUSE

3-1 always set to 0 (interrupt cleared)

After the interrupt has been handled, the routine should terminate with an RTI.

As already mentioned, interrupts may be missed during disk accesses.

If you turn on mouse interrupts without initializing the mouse, the built-in interrupt handler will absorb the interrupts. If you want to handle mouse interrupts yourself, you must write your own interrupt handler and place vectors to it at addresses \$FFFE and \$FFFF in bank-switched RAM. Interrupts will be ignored whenever the \$D000-\$FFFF ROM is switched in.

---

### ***E.6.2 Firmware for Keyboard Interrupts***

The Apple IIc hardware is able to generate an interrupt when a key is pressed. The firmware is able to buffer up to 128 keystrokes, completely transparently, when properly enabled to do so. It saves them in the second half of page 8 of auxiliary memory. After the buffer is full, subsequent keystrokes are ignored. Because interrupts are only generated when keypresses occur, characters generated by the auto-repeat feature are not buffered. They can, however, be read when the buffer is empty.

Once keyboard buffering has been turned on, the next key should be read by calling RDKEY (\$FD0C).

---

#### **Warning**

*Do not call the buffer reading routine directly. Its entry address will not be the same in future versions of the computer.*

---



See Chapter 8.

The special characters **CONTROL-S** (stop list) and **CONTROL-C** (stop Applesoft execution) do not work while keyboard buffering is turned on. A new keystroke, **APPLE-CONTROL-X**, clears the buffer.

---

### ***Using Keyboard Buffering Firmware***

Keyboard buffering is automatically turned on when the serial firmware is placed in terminal mode. Otherwise you must turn it on yourself.

1. Disable processor interrupts (SEI).
2. Set location \$5FA to \$80. This tells the firmware to buffer keystrokes without calling the user's interrupt handler.
3. Set locations \$5FF and \$6FF to \$80. These are pointers to where in the buffer the next keystroke will be stored and where the next will be read from, respectively.
4. Turn on the ACIA for port 2 by setting the low nibble of \$C0AA to the value \$F. For example:

```
LDA $C0AA    ;read port 2 ACIA command register
ORA #$0F     ;set low nibble to $F
STA $C0AA    ;set port 2 ACIA command register
```

If you are using the serial ports at the same time, just set the low bit of \$C0AA to 1. This prevents receiver interrupts from being turned off.

A PR#2 or IN#2 or the equivalent will shut off keyboard interrupts.

5. Enable processor interrupts (CLI).

---

### ***Using Keyboard Interrupts Through Firmware***

Keyboard interrupts are received through the ACIA for port 2. They can be enabled as follows:

1. Disable processor interrupts (SEI).
2. Set location \$5FA to \$C0. This tells the firmware to identify a keystroke interrupt, and to call the user's interrupt handler.
3. Turn on the ACIA for port 2 by setting the low nibble of \$C0AA to the value \$F. For example:

```

LDA $C0AA    ;read port 2 ACIA command register
ORA #$0F     ;set low nibble to $F
STA $C0AA    ;set port 2 ACIA command register

```

#### 4. Enable processor interrupts (CLI).

When the user's interrupt handler is called, it can identify the keyboard as the interrupt source by reading location \$4FA. This is a copy of the ACIA status register at the time of the interrupt. If the interrupt was due to something on the ACIA for port 2, bit 7 is set. If the interrupt was caused by a keystroke, bit 6 is set and bit 5 is unchanged.

After servicing this interrupt, the interrupt handler should clear the interrupt by setting \$4FA to 0.

### ***E.6.3 Using External Interrupts Through Firmware***

Pin 9 of the external disk drive connector (EXTINT) can be used to generate interrupts through the ACIA for port 1. It can be used as a source of interrupts (on a high-to-low transition) if enabled as follows:

1. Disable processor interrupts (SEI).
2. Set location \$5F9 to \$C0. This tells the firmware to identify an external interrupt, and to call the user's interrupt handler.
3. Turn on the ACIA for port 1 by setting the low nibble of \$C09A to the value \$F. For example:

```

LDA $C09A    ;read port 1 ACIA command register
ORA #$0F     ;set low nibble to $F
STA $C09A    ;set port 1 ACIA command register

```

#### 4. Enable processor interrupts (CLI).

When the user's interrupt handler is called, it can identify this interrupt by reading location \$4F9. This is a copy of the ACIA status register at the time of the interrupt. If the interrupt was due to something on the ACIA for port 1, bit 7 is set. If the interrupt was caused by the external interrupt line, bit 6 is clear and bit 5 is unchanged.

After servicing this interrupt, the interrupt handler should clear the interrupt by setting \$4F9 to 0.

---

### ***E.6.4 Firmware for Serial Interrupts***

The Apple IIc hardware is able to generate interrupts both when the ACIA receives data and when it is ready to send data. The built-in interrupt handler responds to incoming data only. The firmware is able to buffer up to 128 incoming bytes of serial data from either serial port. After the buffer is full, data is ignored. Only one port can be buffered at a time.

The following sections assume that the serial port to be buffered is already initialized, as explained in Chapter 8.

---

#### ***Using Serial Buffering Transparently***

Serial buffering is automatically turned on when the serial firmware is placed in terminal mode. Otherwise you must turn it on yourself, as follows:

1. Disable processor interrupts (SEI).
2. Set location \$4FF to \$C1 to buffer port 1, or to \$C2 to buffer port 2.
3. Set locations \$57F and \$67F to 0. These are pointers to the next byte in the buffer to be used and the next character to be read from the buffer, respectively.
4. Turn on the ACIA for the port by setting the low nibble of \$C09A for port 1 or \$C0AA for port 2 to \$D. For example:

```
LDA $C09A    ;read port 1 ACIA command register
AND $F0      ;clear low nibble
ORA #$0D     ;set low nibble to $D
STA $C09A    ;set port 1 ACIA command register
```

The 0 in bit 1 of the command register enables receiver interrupts, thus an interrupt is generated when a byte of data is received.

5. Enable processor interrupts (CLI).

See Chapter 8.

When serial port buffering is thus enabled, normal reads from the serial port firmware fetch data from the buffer rather than directly from the ACIA.

---

### ***Using Serial Interrupts Through Firmware***

It is also possible to use the firmware to call the user interrupt handler whenever a byte of data is read by the ACIA. In this mode buffering is not performed by the firmware.

1. Disable processor interrupts (SEI).
2. Set location \$4FF to a value other than \$C1 or \$C2.
3. Turn on the ACIA for the port by setting the low nibble of \$C09A for port 1 or \$C0AA for port 2 to \$D. For example:

```
LDA $C09A    ;read port 1 ACIA command register
AND $F0      ;clear low nibble
ORA #$0D     ;set low nibble to $D
STA $C09A    ;set port 1 ACIA command register
```

The 0 in bit 1 of the command register enables receiver interrupts, thus an interrupt is generated when a byte of data is received.

4. Enable processor interrupts (CLI).

When a serial port is thus enabled, the user's interrupt handler is called each time the port receives a byte of data. The status byte saved by the firmware (\$4F9 for port 1; \$4FA for port 2) has the high bit set if the interrupt occurred on that port. Bit 3 is set if the interrupt was due to a received byte of data.

The interrupt handler should clear the interrupt by clearing bits 7 and 3 of that port's status byte (\$4F9 for port 1; \$4FA for port 2).

---

### ***Transmitting Serial Data***

The serial firmware does not implement buffering for serial output. Instead it waits for two conditions to be true before transmitting a character:

- The ACIA's transmit register must be ready to accept a character. This is true if bit 4 of the ACIA's status register is 1.
- The device must signal that it is ready to accept data. This is true if bit 5 of the ACIA's status register is 0. Bit 5 is 0 if pin 5 of the port's connector is also 0.

When the serial firmware is active, a change of state on pin 5 of that port generates an interrupt. That interrupt is absorbed, but the data remains in bit 5 of the status register. Interrupts from the ACIA's transmit register are normally disabled.

---

### ***A Loophole in the Firmware***

So that programs can make use of interrupts on the ACIAs without affecting mouse interrupt handling, there is a tiny loophole purposely left in the built-in interrupt handler. If transmit interrupts are enabled on the ACIA—that is, if bits 3, 2, and 0 of the ACIA's command register have the values 0, 1, and 1, respectively—then control is passed to the user's interrupt handler if the interrupt is not intended for the mouse (movement, button, or VBL).

This means that you can write more sophisticated serial interrupt handling routines than the limited firmware space could provide (such as printer spooling). The firmware will still set memory to its standard state, handle mouse interrupts, and restore memory after your routine is finished.

When you receive the interrupt, neither ACIA's status register has been read. You are fully responsible for checking for interrupts on both ACIAs, determining which of the four interrupt sources on each ACIA caused the interrupt, and how to handle them. Refer to the 6551 specification for more details. The built-in firmware itself is an excellent example of how interrupts on the ACIA can be handled.



## E.7 Bypassing the Interrupt Firmware

The following sections give further details on using interrupts on the Apple IIc computer without using the built-in interrupt handler.

### E.7.1 Using Mouse Interrupts Without the Firmware

To use mouse interrupts without the firmware, as mentioned above, you must set your own interrupt vectors. If \$D000-\$FFFF ROM is ever switched in, the built-in interrupt handler will absorb the mouse interrupts. Tables E-1 and E-2 show how to activate and read mouse interrupts without using the firmware. Remember to disable interrupts (SEI) before enabling mouse interrupts, then turn them on when done (CLI).

Table E-1. Activating Mouse Interrupts

To Activate Interrupts On	Enable IOU Access	Select Source	Enable Source	Disable IOU Access
Mouse X (rising edge)	STA \$C079	STA \$C05C	STA \$C059	STA \$C078
Mouse X (falling edge)	STA \$C079	STA \$C05D	STA \$C059	STA \$C078
Mouse Y (rising edge)	STA \$C079	STA \$C05E	STA \$C059	STA \$C078
Mouse Y (falling edge)	STA \$C079	STA \$C05F	STA \$C059	STA \$C078
VBL	STA \$C079		STA \$C05B	STA \$C078

Table E-2. Reading Mouse Interrupts

To Read Interrupts From	Read Direction (A.S.A.P.)	Determine Source	Handle It	Return
Mouse X	LDA \$C066	LDA \$C016 (bit 7 = 1 if true)	...	RTI
Mouse Y	LDA \$C067	LDA \$C017 (bit 7 = 1 if true)	...	RTI
VBL		LDA \$C019 (bit 7 = 1 if true)	...	RTI

The mouse direction data read from \$C066 and \$C067 is guaranteed to be valid for at least 40 microseconds. Average duration is at least 200 microseconds. This means you should read the direction as soon as possible.



---

### ***E.7.2 Using ACIA Interrupts Without the Firmware***

To use ACIA interrupts without the firmware, you must set your own interrupt vectors. If the \$D000-\$FFFF ROM is ever switched in, the built-in interrupt handler will handle the interrupt as determined by certain mode bytes (section E.6.1).

When writing your serial interrupt handler, refer to Figures 11.31 through 11.33 and to the Synertek 6551 ACIA specification. As shown in Chapter 11, the ACIA's have the following connections:

- Port 1:      DSR line connected to the EXTINT line on the external disk port
- DCD line connected to pin 5 of Port 1 connector
- Port 2:      DSR line goes high when a key is pressed
- DCD line connected to pin 5 of Port 2 connector

The ACIA registers have the following addresses:

Port 1:	Data Register	=	\$C098
	Status Register	=	\$C099
	Command Register	=	\$C09A
	Control Register	=	\$C09B
Port 2:	Data Register	=	\$C0A8
	Status Register	=	\$C0A9
	Command Register	=	\$C0AA
	Control Register	=	\$C0AB

# *Apple II Series Differences*

This appendix compares the Apple IIc to the Apple IIe, Apple II Plus, and Apple II.

## F.1 Overview

This appendix does not contain an exhaustive list of differences. However, it does include those differences most likely to affect the accuracy of programs, displays, and instructions created for end users of two or more models from the Apple II Series.

As an overview, the differences between the Apple II series computers can be expressed as a series of equations: this computer equals that one plus or minus certain features.

**Note:** The following *equations* are merely an overview of what each model of Apple II Series is with respect to its predecessor. These equations are in terms of functional equivalence, not strict equality. For example,

Apple IIe = Apple II Plus + Apple Language Card

does not mean there is an actual language card or slot—just that the one machine functions as if it were the other with such a card (with its connector) in a slot.

Apple II Plus	=	II	<ul style="list-style-type: none"> <li>+ Autostart ROM</li> <li>+ Applesoft firmware</li> <li>+ 48K RAM standard</li> <li>- Old Monitor ROM</li> <li>- Integer BASIC firmware</li> </ul>
Apple IIe	=	II Plus	<ul style="list-style-type: none"> <li>+ Apple Language Card (with 16K of RAM)</li> <li>+ 80-column (enhanced) video firmware</li> <li>+ built-in diagnostics</li> <li>+ full ASCII keyboard</li> <li>+ internal power light</li> <li>+ FCC approval</li> <li>+ improved back panel</li> <li>+ 9-pin back panel game connector</li> <li>+ auxiliary slot (with possibility of 80-column text card</li> <li>+ extra 64K RAM)</li> <li>- slot 0</li> </ul>
Apple IIc	=	IIe	<ul style="list-style-type: none"> <li>+ extended 80-column text card</li> <li>+ 80/40 switch</li> <li>+ keyboard switch</li> <li>+ disk light</li> <li>+ disk controller port</li> <li>+ disk drive</li> <li>+ mouse port</li> <li>+ serial printer port</li> <li>+ serial communication port</li> <li>+ built-in port firmware</li> <li>+ video expansion connector</li> <li>- removable cover</li> <li>- slots 1 to 7</li> <li>- auxiliary slot</li> <li>- internal power light</li> <li>- cassette I/O connectors</li> <li>- internal game I/O connector (hence no game <i>output</i>)</li> <li>- auxiliary video pin</li> <li>- monitor cassette support</li> </ul>

---

### ***F.1.1 Type of CPU***

The CPU in the Apple II and II Plus is the 6502. The Apple IIe uses a 6502A. The Apple IIc uses the 65C02; this is a redesigned CMOS CPU that has 27 new instructions, new addressing modes, and for some instructions a differing execution scheme and machine cycle counts (see Appendix A). Programs written for the Apple IIc will run on the earlier machines only if they do not contain instructions unique to the 65C02, or depend on instruction cycle times that differ.

---

### ***F.1.2 Machine Identification***

Identification of Apple II series computers is as follows:

Machine	\$FB83	\$FBC0	\$FB1E
Apple II	\$38		
Apple II Plus	SEA		
Apple IIe	\$06	SEA	
Apple IIc	\$06	\$00	
Apple III in Apple II Emulation Mode	SEA		\$8A

Any future Apple II series computer or ROM release will have different values in these locations. Machine identification routines are available from Apple Vendor Technical Support.

With regard to ProDOS, its MACHID byte, at location \$BF98 on the global page, will have bit 3 set to 0 if the computer is an Apple II, II Plus, IIe, or III, and a 1 if the computer is not one of these machines. In addition, for an Apple IIc, bits 7 and 6 are set to binary 10.

Bits 7 and 6 set to binary 10 indicate that a computer is Apple IIe and IIc compatible, regardless of the value of bit 3.

---

## **F.2 Memory Structure**

This section compares the memory organization of the Apple IIc with that of the Apple II, II Plus, and IIe. These machines differ in RAM space, ROM space, slot or port address space, and hardware page use.

---

### **F.2.1 Amount and Address Ranges of RAM**

The Apple II could have as little as 4K of RAM at the time of purchase, and could be upgraded to as much as 48K of RAM, following a procedure described in the *Apple II Reference Manual*.

The Apple II Plus has 48K of RAM (\$0000 through \$BFFF) as a standard feature. With the addition of an Apple Language Card, a 48K Apple II or II Plus could be expanded to have 64K of RAM.

The Apple IIe has a full 64K of RAM. The top 12K addresses overlap with the ROM addresses \$D000 through \$FFFF. There is an additional bank-switched area of 4K from \$D000 through \$DFFF. This arrangement is equivalent to an Apple II Plus with an Apple Language Card installed. A program selects between the RAM and ROM address spaces and between the \$Dxxx banks by changing soft switches located in memory.

With an Extended 80-Column Text Card installed in its auxiliary slot, an Apple IIe has an additional 64K of RAM available, although no more than half of the 128K of RAM space is available at any given time. Soft switches located in memory control these address space selections.

The RAM in the Apple IIc is equivalent to the RAM in an Apple IIe with an Extended 80-column Card.



---

### ***F.2.2 Amount and Address Ranges of ROM***

The Apple II has 8K of ROM (\$E000 through \$FFFF), and the Apple II Plus has 12K of ROM (\$D000 through \$FFFF). Users can plug their own ROMs into the sockets provided. The on-board (as opposed to slot) ROM address range is from \$D000 through \$FFFF.

The Apple IIe has 16K of ROM, of which it uses 15.75 K (addresses \$C100 through \$FFFF; page \$C0 addresses are for I/O hardware). ROM addresses \$C300 through \$C3FF (normally assigned to the ROM in a card in slot 3) and \$C800 through \$CFFF contain 80-column video firmware; ROM addresses \$C100 through \$C2FF and \$C400 through \$C7FF (normally assigned to the ROM on cards in slots 1, 2, 4, 5, 6 and 7) contain built-in self-test routines.

A soft switch in RAM controls whether the video firmware or slot 3 card ROM is active. Invoking the self-tests with (⌘)-(CONTROL)-(RESET) causes the self-test firmware to take over the slot ROM address spaces.

The Apple IIc ROM also uses the 15.75 K from \$C100 through \$FFFF, and its enhanced video firmware has the same entry point addresses as on the Apple IIe. However, there are only rudimentary built-in self-tests, and these do not pre-empt any port firmware space.

In the Apple IIc, addresses \$C100 through \$CFFF contain I/O and interrupt firmware, addresses \$D000 through \$F7FF contain the Applesoft BASIC Interpreter, and addresses \$F800 through \$FFFF contain the Monitor.

---

### ***F.2.3 Peripheral-Card Memory Spaces***

Each Apple IIc port has up to sixteen peripheral-card I/O space locations in main memory on the hardware page (beginning at location  $\$C0s0 + \$80$  for slot or port  $s$ ), allocated in the standard Apple II series way (that is, beginning at location  $\$C0s0 + \$80$  for each slot  $s$ ).

The peripheral-card ROM space (page  $\$Cs$  for slot  $s$  in the Apple II, II Plus, and IIe) contains the starting and entry-point addresses for port  $s$ , but port routines are not limited to their allocated  $\$Cs$  pages.

The 2K-byte expansion ROM space from  $\$C800$  to  $\$CFFF$  in the Apple IIc is used by the enhanced video firmware and miscellaneous I/O and memory-transfer routines.

The 128 bytes of peripheral-card RAM space or *scratch-pad RAM* (64 screen holes in main memory and their equivalent addresses in auxiliary memory) are reserved for use by the built-in firmware. It is extremely important for the correct operation of Apple IIc firmware that these locations not be altered by software except for the specific purposes described in Chapters 7, 8, and 9, and in Appendix E.

---

### ***F.2.4 Hardware Addresses***

The hardware page (the addresses from  $\$C000$  through  $\$C0FF$ ) controls memory selection and input/output hardware characteristics. All input and output (except video output) takes place at one or more hardware page addresses. For the sake of simplicity, this section presents only a general comparison between the Apple IIc on the one hand, and the Apple II, II Plus, and IIe on the other, with respect to most hardware page uses. However, for many characteristics, the Apple IIe and IIc work one way, while the Apple II and II Plus work another.

---

#### ***\$C000 to \$C00F***

On all Apple II series computers, reading any one of these addresses reads the keyboard data and strobe. On the Apple IIe and IIc, writing to each of these addresses turns memory and display switches on and off. Writing to addresses \$C006, \$C007, \$C00A, and \$C00B performs ROM selection on the Apple IIe. Writing to these four addresses is reserved on the Apple IIc.

For reading the keyboard, use \$C000; reserve \$C001 through \$C00F.

---

#### ***\$C010 to \$C01F***

On all Apple II series computers, writing to any one of these addresses clears the keyboard strobe. On the Apple IIe and IIc, reading each of these addresses checks the status of a memory or display switch, or the any-key-down flag.

For clearing the keyboard strobe, use \$C010; reserve \$C011 through \$C01F.

Reading \$C015 checks the SLOTCXROM switch on the Apple IIe, but it resets the X-movement interrupt (XINT) on the Apple IIc. Similarly, reading \$C017 checks the SLOTC3ROM switch on the Apple IIe, but it resets the Y-movement interrupt (YINT) on the Apple IIc.

Reading \$C019 checks the current state of vertical blanking (VBL) on the Apple IIe, but it resets the latched vertical blanking interrupt (VBLINT) on the Apple IIc.

---

***\$C020 to \$C02F***

On the Apple II, II Plus, and IIe, reading any address \$C02x toggles the cassette output signal. On the Apple IIc, both reading from and writing to these locations are reserved.

---

***\$C030 to \$C03F***

On all Apple II series computers, reading an address of the form \$C03x toggles the speaker. For full Apple II series compatibility, toggle the speaker using \$C030, and reserve \$C031 through \$C03F.

On the Apple IIc, writing to these addresses is explicitly reserved.

---

***\$C040 to \$C04F***

On the Apple II, II Plus, and IIe, reading any address of the form \$C04x triggers the Utility Strobe. The Apple IIc has no Utility Strobe.

On the Apple IIc, addresses \$C044 through \$C047 are explicitly reserved, and reading or writing any address from \$C048 through \$C04F resets both the X and Y interrupts (XINT and YINT).

---

***\$C050 to \$C05F***

Addresses \$C050 through \$C057 work the same on the Apple IIc as on the Apple IIe: they turn the TEXT, MIXED, PAGE2 and HIRES switches on and off.

On the Apple IIe, addresses \$C058 through \$C05F turn the annunciator outputs on and off. On an Apple IIe with a revision B main logic board, an Apple Extended 80-Column Text Card, and a jumper installed on the card, reading locations \$C05E and \$C05F set and clear double-high-resolution display mode.

On the Apple IIc, if the IOUDIS switch is on, both reading from and writing to addresses \$C058 through \$C05D are reserved, and addresses \$C05E and \$C05F set and clear double-high-resolution display (as on the Apple IIe equipped as described in the preceding paragraph). If the IOUDIS switch is off, then addresses \$C058 through \$C05F control various characteristics of mouse and vertical blanking interrupts (Table 9-2).

---

#### ***\$C060 to \$C06F***

On the Apple IIc, writing to any address of the form \$C06x is reserved, and reading addresses \$C068 through \$C06F is reserved.

Reading addresses \$C061 and \$C062 is the same as on the Apple IIe (switch inputs and Apple keys). Reading addresses \$C064 and \$C065 is the same as on all other Apple II series computers (analog inputs 0 and 1).

On the Apple IIc, address \$C063 bit 7 is 1 if the mouse switch is not pressed, and 0 if it is pressed, so that software looking for the *shift-key mod* (used on Apple II, II Plus, and IIe with some text cards) will find it and display lowercase correctly. If by chance the mouse button is pressed when the software checks location \$C063, it will appear that the shift-key mod is not present.

On the Apple IIc, address \$C060 is used for reading the state of the 80/40 switch; on the Apple II, II Plus, and IIe, this address is for reading cassette input.

The Apple IIc has two, rather than four, analog (*paddle*) inputs. Addresses \$C066 and \$C067 are used for reading the mouse X and Y direction bits.

---

#### ***\$C070 to \$C07F***

On the Apple II, II Plus, and IIe, reading from or writing to any address of the form \$C07x triggered the (analog input) paddle timers.

On the Apple IIc, only address \$C070 is to be used for that one function. Addresses \$C071 through \$C07D are explicitly reserved. The results of reading from or writing to addresses \$C07E and \$C07F are described in Table 5-8.

---

#### ***\$C080 to \$C08F***

On the Apple IIe and IIc, accessing addresses in this range selects different combinations of bank-switched memory banks. However, addresses \$C084 through \$C087 duplicate the functions of the four addresses preceding them, and addresses \$C08C through \$C08F do also. These eight addresses are explicitly reserved on the Apple IIc.

---

#### ***\$C090 to \$C0FF***

On the Apple II, II Plus, and IIe, each group of 16 addresses of the form \$C080 + \$s0 is allocated to an interface card (if present) in slot s.

On the Apple IIc, addresses corresponding to slots 1, 2, 3, 4 and 6 are allocated to a serial interface card, communication interface card, 80-column text card, mouse interface card, and disk controller card, respectively. All other addresses in this range are reserved.



---

### ***F.2.5 Monitors***

The older models of the Apple II and Apple II Plus included a different version of the System Monitor from the one built into more recent models (and the Apple IIe and IIc). The older version, called the Monitor ROM, had the same standard I/O subroutines as the newer Autostart ROM, but a few of their features were different: for example, there were no arrow keys for vertical cursor motion.

When you start the Apple IIc with a DOS or BASICS disk and it loads Integer BASIC into the bank-switched area in RAM, it loads the old Monitor along with it. When you type `INT` from Applesoft to activate Integer BASIC, you also activate this copy of the old Monitor, which remains active until you either type `FP` to switch back to Applesoft, which uses the new Monitor in ROM, or activate the 80-column firmware.

---

## **F.3 I/O in General**

Apple IIc I/O is different from I/O on the Apple II, II Plus, and IIe in three important respects: the possibility of direct memory access (DMA) transfers, the presence or absence of slots, and the presence or absence of built-in interrupt handling.

---

### **F.3.1 DMA Transfers**

The Apple II, II Plus, and IIe allow DMA transfers, because both the address and the data bus are available at the slots. No true DMA transfer is possible with the Apple IIc because neither bus is available at any of the back-panel connectors.

---

### **F.3.2 Slots Versus Ports**

The Apple II and II Plus have eight identical slots; the Apple IIe has seven identical slots plus a 60-pin auxiliary slot for video, add-on memory and test cards. The Apple IIc has no slots; instead, it has built-in hardware and firmware that are functional equivalents of slots with cards in them (and back-panel connectors). These are called **ports** on the Apple IIc.

---

### **F.3.3 Interrupts**

The Apple IIc is the first computer in the Apple II Series to have built-in interrupt-handling capabilities.

Interrupts on the Apple IIc are described in Appendix E.

## **F.4 Keyboard**

Both keyboard layout and character sets vary in the Apple II series computers. The major keyboard difference in the Apple II Series is that the Apple IIe and IIc have full ASCII keyboards, while the Apple II and II Plus do not.

### **F.4.1 Keys**

The Apple II and II Plus have identical 52-key keyboards. The Apple IIe and Apple IIc keyboards have the same 63-key full ASCII keyboard layout, with new and repositioned keys and characters as compared to the Apple II and II Plus. While the Apple II and II Plus have a (REPT) key, the IIe and IIc have an auto-repeat feature built into each character key.

Some Apple II and Apple II Plus machines have a slide switch inside the case, under the keyboard edge of the cover, for selecting whether or not (RESET) works without (CONTROL). On the Apple IIe and Apple IIc, there is no choice: (CONTROL)-(RESET) works, and (RESET) alone does not.

The Apple IIc and IIe have an (⌘) and a (⌘) key; the Apple II and II Plus do not have these two keys.

The captions on several keys—(ESC), (TAB), (CONTROL), (SHIFT), (CAPS-LOCK), (DELETE), (RETURN), and (RESET)—can vary: on the Apple II and II Plus some are abbreviated or missing; on the Apple IIc all keycaps are lowercase italic; on international models, some captions are replaced by symbols (Appendix G).

The Apple IIc has two switches that the other models do not have. One switch is for changing between 40-column and 80-column display, the other is for selecting keyboard layout (Sholes versus Dvorak on USA models), or both keyboard layout and character set (on international models).

The position of the power-on light differs on the Apple II and II Plus, Apple IIe, and Apple IIc. The Apple IIc has a disk-use light as well.

---

#### ***F.4.2 Character Sets***

The Apple II and II Plus keyboard character sets are the same. They are described in the *Apple II Reference Manual*.

The Apple IIe and Apple IIc keyboard character sets are the same: full ASCII. The standard (Sholes) layout and key assignments are described in the *Apple IIe Reference Manual*. The Dvorak layout and key assignments are described in Chapter 4 and Appendix G of this manual.

To change between the two available keyboard layouts requires modification to the main logic board on the Apple IIe, but only toggling of the keyboard switch on the Apple IIc.

Apple Computer, Inc. manufactures fully localized models (power supply and character sets) of both the Apple IIe and the Apple IIc. However, there are minor variations in keyboard layout, even among early and late productions models of the same machine. For further details, refer to Appendix G of this manual or to the Apple IIe *Supplement to the Owner's Manual*.

---

## **F.5 Speaker**

The Apple IIc has two speaker features that the three previous models do not have. They are a two-channel, but monaural, mini-phone jack for headphones—which disconnects the internal speaker when something is plugged into it—and a volume control.

---

## **F.6 Video Display**

This section discusses the general differences between Apple IIc video display capabilities and those of the other computers in the series. Note however that as new ROMs become available for the Apple IIe, many differences between these two machines will vanish.

---

### **F.6.1 Character Sets**

The Apple II and II Plus display only uppercase characters, but they display them in three ways: normal, inverse, and flashing. The Apple IIc and IIe can display uppercase characters in all three ways, and they can display lowercase characters in the normal way. This combination is called the **primary character set**.

The Apple IIc and IIe have another character set, called the **alternate character set**, that displays a full set of normal and inverse uppercase and lowercase characters, but can't display flashing characters. The primary and alternate character sets are described in Chapter 5. You can switch character sets at any time by means of the ALTCHAR soft switch, also described in Chapter 5.

Flashing display must not be used with the enhanced video firmware active. Use it in 40-column mode with the enhanced video firmware turned off; otherwise, strange displays may result, such as MouseText characters appearing in place of uppercase letters.

To be compatible with some software, you have to switch the Apple IIc keyboard to uppercase by pressing **(CAPS LOCK)**.

---

### ***F.6.2 MouseText***

MouseText characters (Chapter 5) are available on every Apple IIc, and on any Apple IIe that has had its ROMs appropriately upgraded, if necessary.

---

### ***F.6.3 Vertical Blanking***

A signal called **vertical blanking** indicates when a display device should stop projecting dots until the display mechanism returns from the bottom of the screen to the top to make another pass. During this interval, a program can make changes to display memory pages, and thus provide a smooth, flicker-free transition to a new display.

On the Apple IIe, vertical blanking (VBL) is a signal whose level must be polled. (VBL is not available to software on the Apple II or II Plus.) On the Apple IIc, vertical blanking is an interrupt (VBLINT) that occurs on the trailing edge of the active-low VBL signal. Programs intended to run on all Apple II series computers must take this difference into account.

---

### ***F.6.4 Display Modes***

All models have 40-column text mode, low-resolution graphics mode, high-resolution graphics mode, and mixed graphics and text modes. The Apple IIe (revision B motherboard) with an Apple Extended 80-Column Text Card, and the Apple IIc have double-high-resolution graphics mode also.



---

## **F.7 Disk I/O**

The Apple II, II Plus, and IIe can support up to six (four is the recommended maximum) disk drives attached in controller cards plugged into slots 6, 5, and 4. The Apple IIc supports up to two disk drives: its built-in drive (treated as slot 6 drive 1), and one external disk drive (treated as slot 6 drive 2; also treated as slot 7 drive 1 under ProDOS) for external-drive startup purposes.

---

## **F.8 Serial I/O**

The Apple IIc serial ports (ports 1 and 2) are similar to Super Serial Cards installed in slots 1 and 2 of an Apple IIe. The serial port commands are a slightly modified subset of Super Serial Card commands. This subset includes all the commands supported by the earlier Apple Serial Interface Card and Communication Card.

---

### **F.8.1 Serial Ports Versus Serial Cards**

There are several important differences between Apple IIc serial ports and other Apple II series computers with serial cards installed in them.

Apple IIc serial ports have no switches. Instead, initial values are moved from firmware locations into auxiliary memory when the power is turned on. Changes made to these values in auxiliary memory remain in effect until the power is turned off. Pressing **⌘-CONTROL-RESET** does not change them.

When the port itself is turned on (with an IN or PR command), the initial values in auxiliary memory are placed in the main memory screen holes assigned to the port. These characteristics can be changed by the port commands. The changed characteristics remain in effect until the port is turned off and then on again (with PR and IN commands).

The command syntax for the Apple IIc ports also differs from the syntax for serial cards. A separate command character, **CONTROL-A** or **CONTROL-I**, must precede each individual port command, whereas several commands to a serial card can be strung together between the command character and a carriage return character.

The letters used for some of the commands have been changed from those used with the Super Serial Card (such as *S* instead of *B* for sending a BREAK signal). Each serial port command letter is unique to simplify command interpretation.

Changing the command character from **CONTROL-A** to **CONTROL-I**, or vice versa, makes the Super Serial Card change from communication mode to printer mode and back; this is not the case with Apple IIc serial ports. With the Apple IIc, use the *System Utilities Disk* to change modes.

Super Serial Card commands support several functions that Apple IIc serial port commands don't support: masking incoming line feed after carriage return; translating incoming characters, such as changing lowercase to uppercase (for the benefit of the Apple II or II Plus); delaying after sending carriage return, line feed, or form feed; ignoring keyboard input, and so on.

Following a **CONTROL-I nnnN** command, the Apple IIc automatically generates carriage return after *nnn* characters; with the Super Serial Card, you need to turn this on with **CONTROL-I C**.

---

### ***F.8.2 Serial I/O Buffers***

The communication port firmware uses auxiliary memory page 8 as an input and output buffer. By doing so, the firmware can keep up with higher baud rates. It can also *hide* data from the Monitor, Applesoft, and other system software.

Programs written for the Apple IIe or IIc can, of course, store information in auxiliary memory page 8. However, such information will be destroyed when the communication port is activated.

---

## **F.9 Mouse and Hand Controls**

The DB-9 back-panel connector on the Apple IIc is used for both the mouse and hand controls. On the Apple IIc, the DB-9 connector supports hand controls only. On the Apple IIe, the mouse must use the connector on the interface card.

---

### **F.9.1 Mouse Input**

The Apple IIc provides built-in firmware support for a mouse connected to the DB-9 mouse and hand control connector. Apple IIc mouse support includes mouse movement and button interrupts (and vertical blanking interrupts for synchronization with the display); Apple IIe mouse support relies on polling VBL instead of vertical blanking interrupts.

As a result of how interrupts are handled on the two machines, the mouse firmware routine calls function somewhat differently for the Apple IIc and Apple IIe. However, using the calls in the manner described in Chapter 9 ensures mouse support compatibility between the two machines.

The ratio of mouse movement to cursor movement is different on the Apple IIc than it is on the Apple IIe.

---

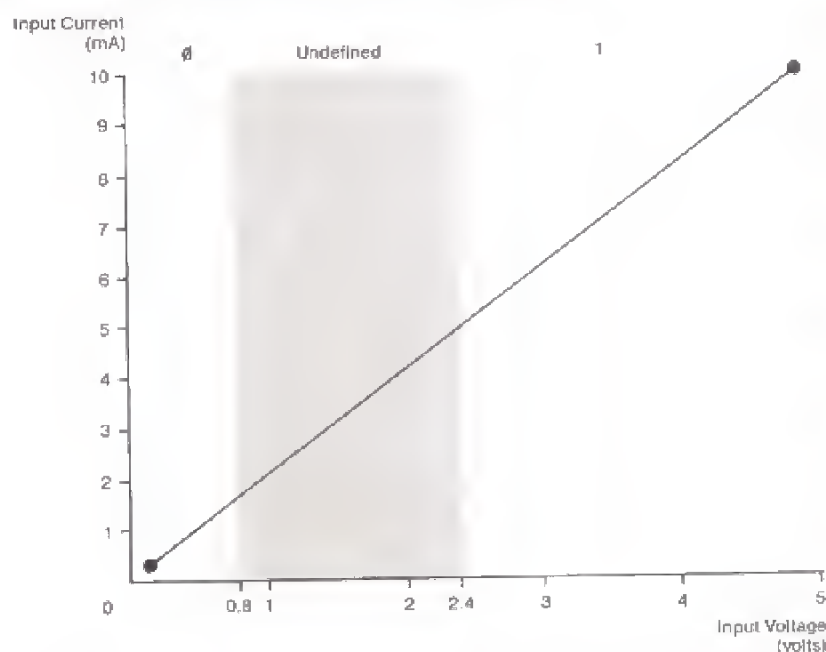
### **F.9.2 Hand Control Input and Output**

The Apple II, II Plus, and IIe have a 16-pin game I/O connector inside the case that supports three switch inputs, four analog (paddle) inputs, and four annunciator outputs. The Apple IIe and Apple IIc have a DB-9 back-panel connector that supports the three switch inputs and two paddle inputs (plus two more on the internal GAME I/O connector of the Apple II, II Plus, and IIe).

The Apple IIc does not support the four annunciator outputs.

The voltage-current curve for hand controls differs for the Apple IIc compared with that of the Apple II, II Plus, and IIe. Compare Figure F-1 with Figure 11-42. This was done so the hardware would support identifiable mouse and hand control signals using the same circuits.

**Figure F-1.** Apple II, II Plus, and IIe Hand Control Signals



The paddle timing circuit on the Apple II Plus is slightly different than the one on the Apple IIe and IIc. On the Apple IIe and IIc the 100 ohm fixed resistor is between the NE556 discharge lead and the capacitor; the variable resistor in the paddle is connected directly to the capacitor. On the Apple II Plus, the capacitor is connected directly to the discharge lead, and the fixed resistor is in series with the paddle resistor.

## **F.10 Cassette I/O**

The Apple II, II Plus, and IIe all have cassette input and output jacks, memory locations, and monitor support. The Apple IIc does not.

## **F.11 Hardware**

Besides the different microprocessors used in various models in the Apple II series (section F.1.1), there are important differences in power specifications and custom chips.

---

### **F.11.1 Power**

The power supplies for the Apple II, II Plus, and IIe are essentially the same. The floor transformer and voltage converter for the Apple IIc have smaller capacity for current and heat dissipation. Therefore, it is important to observe the load limits specified in each of the reference manuals.

---

### **F.11.2 Custom Chips**

The Apple IIe custom chips (Memory Management Unit and Input/Output Unit) replaced dozens of Apple II Plus chips, and added the functionality of dozens more. The Apple IIc has custom MMU and IOU chips, too, but they represent different *bonding options*, and so their pin assignments are not compatible.

In addition, the Apple IIc has a custom General Logic Unit (GLU), Timing Generator (TMG), and Disk Controller Unit (IWM). The Apple IIc has two hybrid units (AUD and VID) for audio and video amplification.

# ***USA and International Models***



This appendix repeats some of the keyboard information given in Chapter 4 for the two USA keyboard layouts for easy comparison with the other layouts available. Following these there is a composite table of the ASCII codes and the characters associated with them on all the models discussed.

## G.1 Keyboard Layouts and Codes

Each of the following subsections has a keyboard illustration and a table of the codes that result from the possible keystrokes. Note, however, that Table G-1 is the basic table of keystrokes and their codes. For simplicity, subsequent tables (up to Table G-6) list only the keystrokes and codes that differ from those in Table G-1.

For example, pressing the (A) key produces *a* (hexadecimal 61); pressing (SHIFT)-(A) produces uppercase *A* (hexadecimal 41); pressing (CONTROL)-(A) or (CONTROL)-(SHIFT)-(A) produces *SOH* (the ASCII Start Of Header control character, hexadecimal 01). You can tell that this key has the same effect on all keyboards, from the fact that nothing appears in Tables G-2 through G-7 for that key.

A quick way to find out which characters in the ASCII set change on international keyboards is to check Table G-7. In fact, only a few of them change. The pairing of characters on keys varies more.

**Note:** On all but the French and Italian keyboards, **(CAPS-LOCK)** affects only keys that can produce both lowercase letters (with or without an accent) and their uppercase equivalents. With these keys, **(CAPS-LOCK)** down is equivalent to holding down **(SHIFT)**, resulting in uppercase instead of lowercase. If a key produces only a lowercase version of an accented letter, then **(CAPS-LOCK)** does not affect it.

On the French and Italian keyboards, **(CAPS-LOCK)** shifts all the keys. Furthermore, on the French keyboard, when **(CAPS-LOCK)** is down, the **(SHIFT)** key undoes the shifting.

**Note:** The shapes and arrangement of keys in Figures G-1 and G-2 follow the ANSI (American National Standards Institute) standard, which is used mainly in North and South America. The shapes and arrangement of keys in Figure G-3 follows the ISO (International Standards Organization) standard used in Europe and elsewhere.

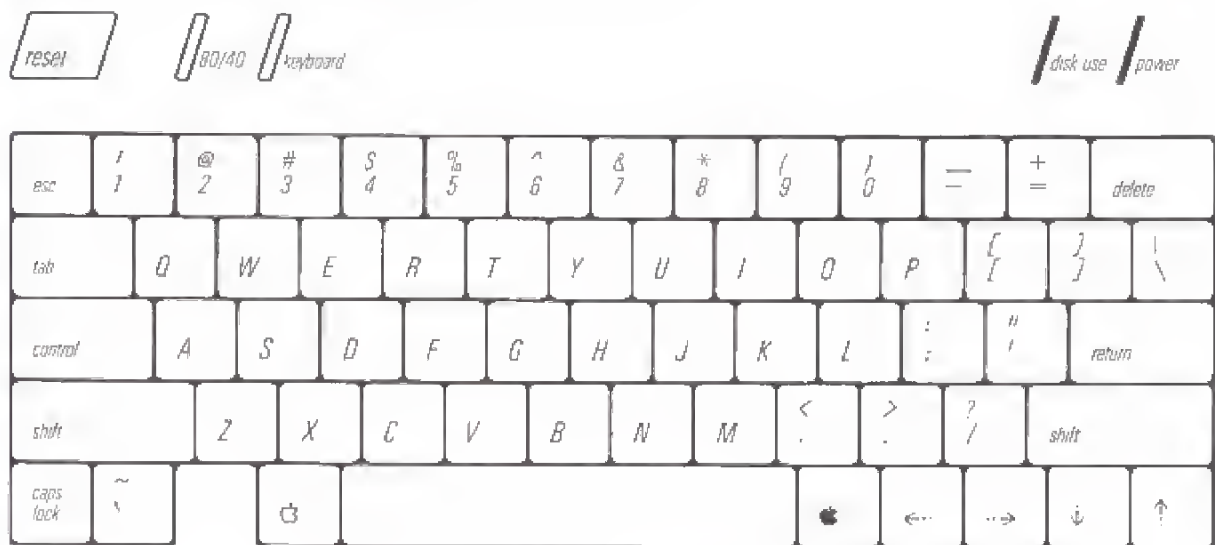
The only differences between the ANSI and ISO versions of the USA keyboard are

- The shapes of three keys: the left **(SHIFT)** key, **(CAPS-LOCK)**, and **(RETURN)**.
- The resulting repositioning of two keys **(|)** and **(=)** in Figures G-1 and G-3.
- For some countries, there are arrow symbols on **(TAB)**, **(CAPS-LOCK)**, **(RETURN)**, and the two **(SHIFT)** keys (as shown in Figure G-3).

### G.1.1 USA Standard (Sholes) Keyboard

Figure G-1 shows the Standard (Sholes) keyboard as it is laid out for USA models of the Apple IIc with the keyboard switch up. Table G-1 lists the ASCII codes resulting from all simple and combination keystrokes on this keyboard.

Figure G-1. USA Standard or Sholes Keyboard (Keyboard Switch Up)



**Table G-1. Keys and ASCII Codes.** Codes are shown here in hexadecimal; to find the decimal equivalents, use Table G-7.

Key	Key Alone Hex	Char	CONTROL + Key Hex	Char	SHIFT + Key Hex	Char	Both + Key Hex	Char
DELETE	7F	DEL	7F	DEL	7F	DEL	7F	DEL
←	08	BS	08	BS	08	BS	08	BS
TAB	09	HT	09	HT	09	HT	09	HT
␣	0A	LF	0A	LF	0A	LF	0A	LF
␣	0B	VT	0B	VT	0B	VT	0B	VT
RETURN	0D	CR	0D	CR	0D	CR	0D	CR
→	15	NAK	15	NAK	15	NAK	15	NAK
ESC	1B	ESC	1B	ESC	1B	ESC	1B	ESC
SPACE	20	SP	20	SP	20	SP	20	SP
·	27	·	27	·	22	·	22	·
· <	2C	·	2C	·	3C	<	3C	<
·	2D	·	1F	US	5F	·	1F	US
· >	2E	·	2E	·	3E	>	3E	>
/ ?	2F	/	2F	/	3F	?	3F	?
0	30	0	30	0	29	)	29	)
1	31	1	31	1	21	!	21	!
2 @	32	2	00	NUL	40	@	00	NUL
3 #	33	3	33	3	23	#	23	#
4 \$	34	4	34	4	24	\$	24	\$
5 %	35	5	35	5	25	%	25	%
6 ^	36	6	1E	RS	5E	·	1E	RS
7 &	37	7	37	7	26	&	26	&
8 *	38	8	38	8	2A	·	2A	·
9 (	39	9	39	9	28	(	28	(
·	3B	·	3B	·	3A	:	3A	:
= +	3D	=	3D	=	2B	+	2B	+
[ \	5B	[	1B	ESC	7B	·	1B	ESC
] ;	5C	]	1C	FS	7C	·	1C	FS
·	5D	·	1D	GS	7D	·	1D	GS
·	60	·	60	·	7E	·	7E	·

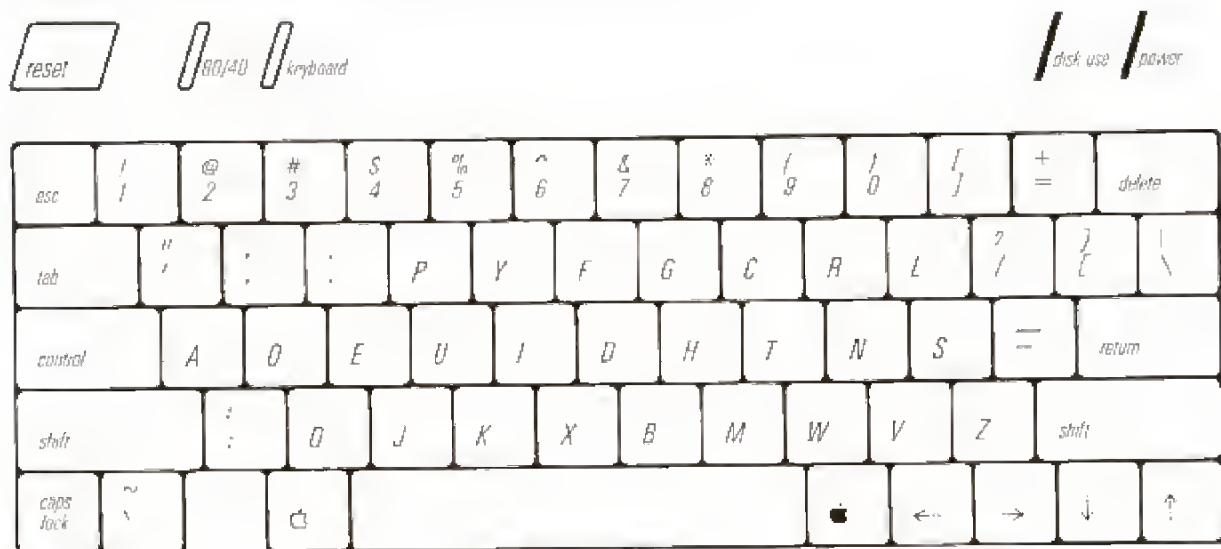
**Table G-1—Continued.** Keys and ASCII Codes. Codes are shown here in hexadecimal; to find the decimal equivalents, use Table G-7.

Key	Key Alone Hex	Char	CONTROL + Key Hex	Char	SHIFT + Key Hex	Char	Both + Key Hex	Char
A	61	a	01	SOH	41	A	01	SOH
B	62	b	02	STX	42	B	02	STX
C	63	c	03	ETX	43	C	03	ETX
D	64	d	04	EOT	44	D	04	EOT
E	65	e	05	ENQ	45	E	05	ENQ
F	66	f	06	ACK	46	F	06	ACK
G	67	g	07	BEL	47	G	07	BEL
H	68	h	08	BS	48	H	08	BS
I	69	i	09	HT	49	I	09	HT
J	6A	j	0A	LF	4A	J	0A	LF
K	6B	k	0B	VT	4B	K	0B	VT
L	6C	l	0C	FF	4C	L	0C	FF
M	6D	m	0D	CR	4D	M	0D	CR
N	6E	n	0E	SO	4E	N	0E	SO
O	6F	o	0F	SI	4F	O	0F	SI
P	70	p	10	DLE	50	P	10	DLE
Q	71	q	11	DC1	51	Q	11	DC1
R	72	r	12	DC2	52	R	12	DC2
S	73	s	13	DC3	53	S	13	DC3
T	74	t	14	DC4	54	T	14	DC4
U	75	u	15	NAK	55	U	15	NAK
V	76	v	16	SYN	56	V	16	SYN
W	77	w	17	ETB	57	W	17	ETB
X	78	x	18	CAN	58	X	18	CAN
Y	79	y	19	EM	59	Y	19	EM
Z	7A	z	1A	SUB	5A	Z	1A	SUB

### G.1.2 USA Simplified (Dvorak) Keyboard

Figure G-2 shows the Dvorak layout of the USA keyboard. Characters are paired up on keys in exactly the same way as on the USA Standard keyboard; only individual key positions are changed. In fact, you can change the keycap arrangement to match Figure G-2, lock the keyboard switch in its down position, and have a working Dvorak keyboard. All keystrokes produce the same ASCII codes as those shown in Table G-1.

Figure G.2. USA Simplified or Dvorak Keyboard (Keyboard Switch Down)

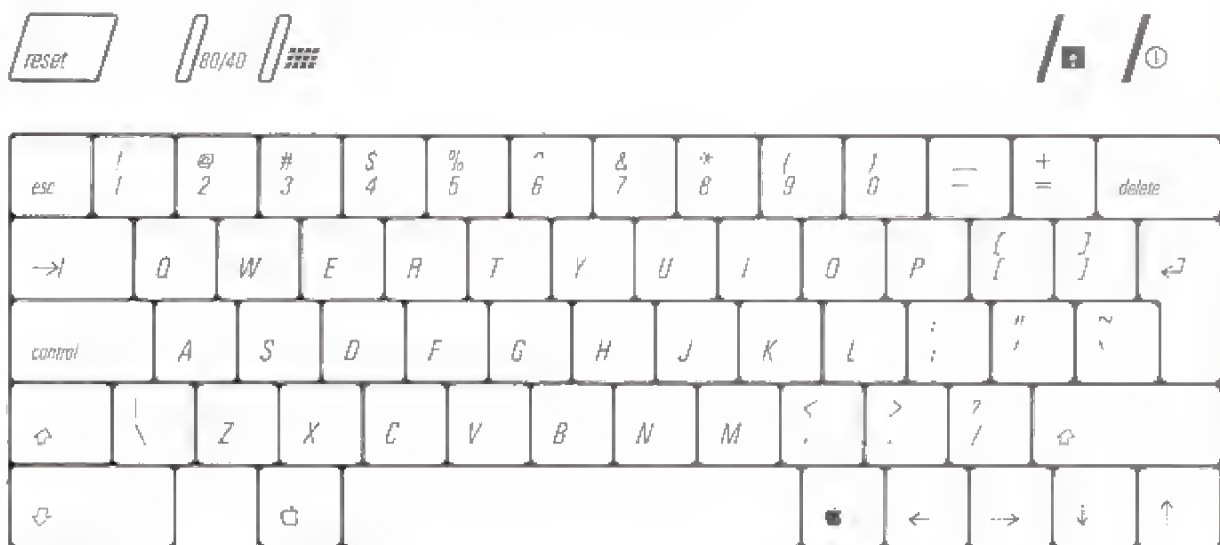




### G.1.3 ISO Layout of USA Keyboard

Figure G-3 shows the layout of the keyboard of all ISO European keyboards (except the Italian keyboard) when the keyboard switch is up. All keystrokes produce the same ASCII codes as those shown in Table G-1.

Figure G-3. ISO Version of USA Standard Keyboard (Keyboard Switch Up)



### G.1.4 English Keyboard

With the keyboard switch up, the English model of the Apple IIc keyboard layout is as shown in Figure G-3, and keystrokes produce the ASCII codes shown in Table G-1.

With the keyboard switch down, the English model keyboard layout is as shown in Figure G-4. The change in ASCII code production (from that in Table G-1) is shown in Table G-2.

The only changed character is the substitution of the British pound-sterling symbol (£) for the cross-hatch symbol (#) on the shifted 3-key.

Figure G-4. English Keyboard (Keyboard Switch Down)

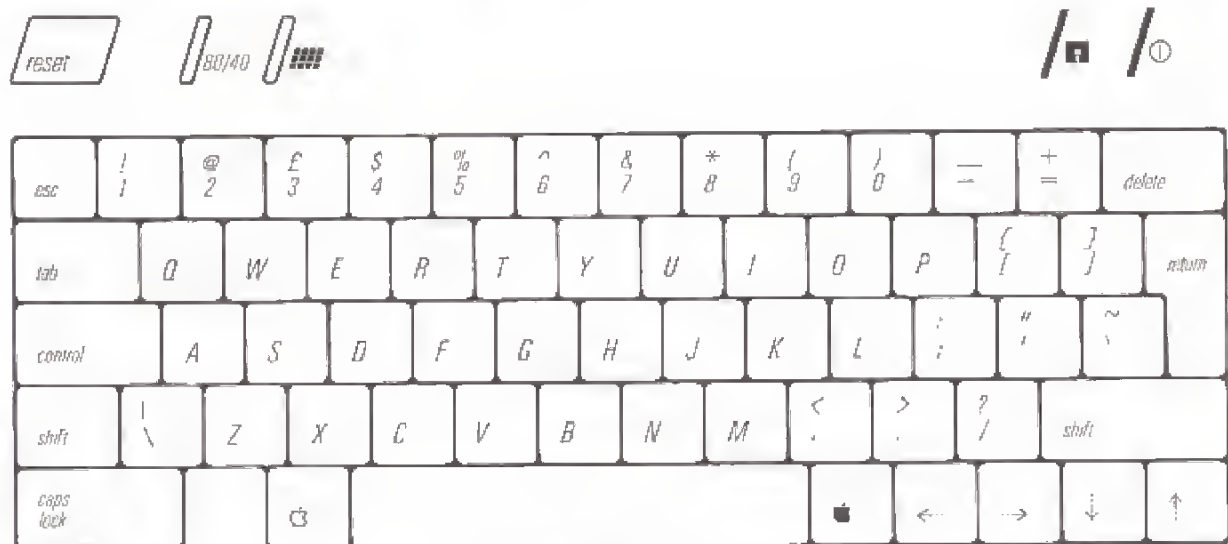


Table G-2. English Keyboard Code Differences From Table G-1

Key	Key Alone Hex	Char	CONTROL + Key Hex	Char	SHIFT + Key Hex	Char	Both + Key Hex	Char
3£	33	3	33	3	23	£	23	#

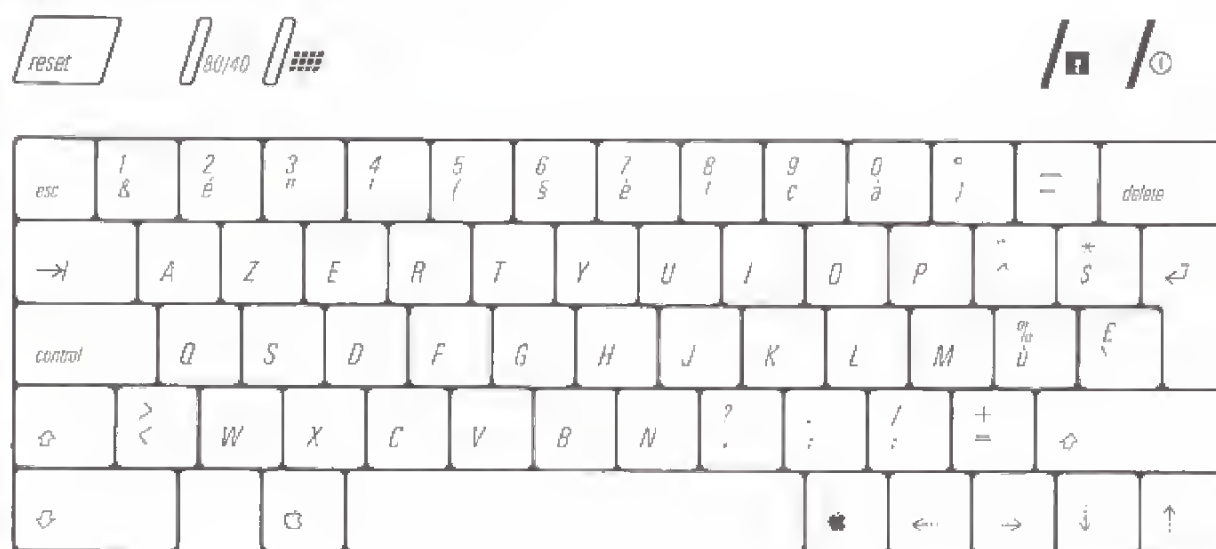
### G.1.5 French and Canadian Keyboards

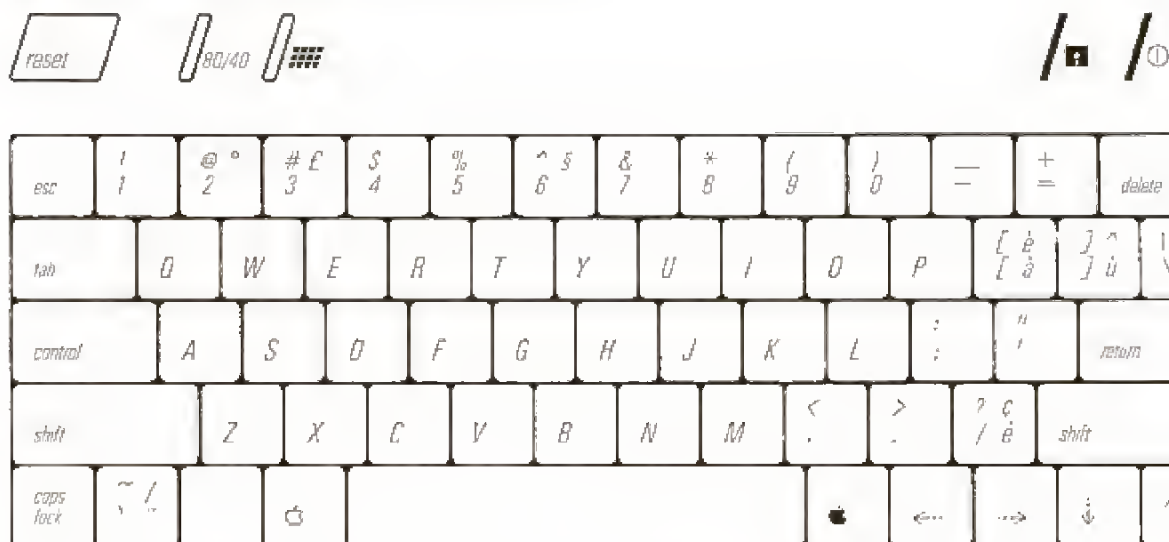
With the keyboard switch up, the French model of the Apple IIc keyboard layout is as shown in Figure G-3, and the Canadian is as shown in Figure G-1. On both models, keystrokes produce the ASCII codes shown in Table G-1.

**Note:** On the French keyboard, **(CAPS-LOCK)** shifts to the upper characters on all keys. With **(CAPS-LOCK)** on, **(SHIFT)** "unshifts" to the lower character on any key pressed with it.

With the keyboard switch down, the French model keyboard layout is as shown in Figure G-5, and the Canadian model keyboard layout is as shown in Figure G-6. The changes in ASCII code production (from that in Table G-1) are shown in Table G-3.

Figure G-5. French Keyboard (Keyboard Switch Down)





**Table G-3. French and Canadian Keyboard Code Differences From Table G-1**

Key	Key Alone Hex	Char	CONTROL Hex	+ Key Char	SHIFT + Key Hex	+ Key Char	Both + Key Hex	+ Key Char
&1	26	&	26	&	31	1	31	1
é2	7B	é	7B	é	32	2	32	2
"3	22	"	22	"	33	3	33	3
'4	27	'	27	'	34	4	34	4
(5	28	(	28	(	35	5	35	5
\$6	5D	\$	1D	GS	36	6	1D	GS
è7	7D	è	7D	è	37	7	37	7
!8	21	!	21	!	38	8	38	8
ç9	5C	ç	1C	FS	39	9	1C	FS
à0	40	à	00	NUL	30	0	00	NUL
)°	29	)	1B	ESC	5B	°	1B	ESC
--	5E	-	1E	RS	7E	-	1E	RS
\$°	24	\$	24	\$	2A	°	2A	°
ù%	7C	ù	7C	ù	25	%	25	%
'£	60	'	60	'	23	£	23	£
<>	3C	<	3C	<	3E	>	3E	>
/?	2C	.	2C	.	3F	?	3F	?
:/	3B	:	3B	:	2E	/	2E	/
	3A	:	3A	:	2F	/	2F	/

With the keyboard switch down, the German model keyboard layout is as shown in Figure G-7. The change in ASCII code production (from that in Table G-1) is shown in Table G-4.

[illegible]

Key	Key Alone Hex	Char	CONTROL + Key Hex	Char	SHIFT + Key Hex	Char	Both + Key Hex	Char
2"	32	2	32	2	22	"	22	"
3\$	33	3	00	NUL	40	\$	00	NUL
6&	36	6	36	6	26	&	26	&
7/	37	7	37	7	2F	/	2F	/
8(	38	8	38	8	28	(	28	(
9)	39	9	39	9	29	)	29	)
0=	30	0	30	0	3D	=	3D	=
0 ?	7E	0	7E	0	3F	?	3F	?
Ü	7D	Ü	1D	GS	5D	Ü	1D	GS
+ *	2B	+	2B	+	2A	*	2A	*
Ö	7C	Ö	1C	FS	5C	Ö	1C	FS
Ä	7B	Ä	1B	ESC	5B	Ä	1B	ESC
#	23	#	1E	RS	5E	#	1E	RS
<>	3C	<	3C	<	3E	>	3E	>
.	2C	.	2C	.	3B	:	3B	:
:	2E	:	2E	:	3A	:	3A	:

### G.1.7 Italian Keyboard

With the keyboard switch down, the Italian model keyboard layout is as shown in Figure G-8. The change in ASCII code production (from that in Table G-1) is shown in Table G-5.

With the keyboard switch up, the Italian model keyboard produces exactly the same ASCII codes for each key, but what is displayed differs for the ten characters shown in Table G-5 or Table G-7.

Figure G-8. Italian Keyboard (Keyboard Switch Down)

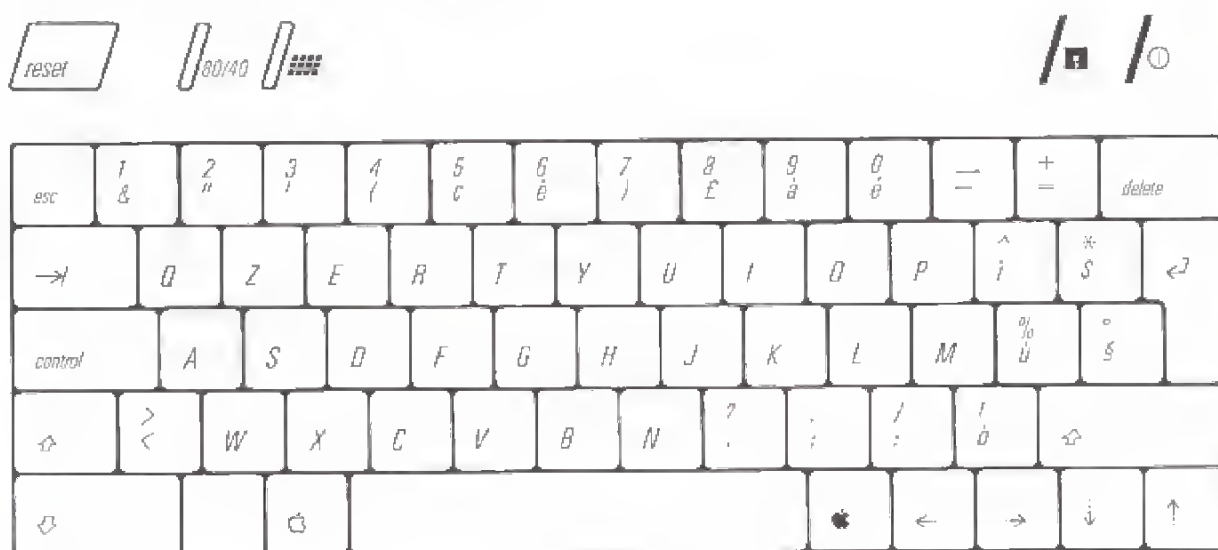




Table G-5. Italian Keyboard Code Differences From Table G-1

Key	Key Alone Hex	Char	CONTROL + Key Hex	Char	SHIFT + Key Hex	Char	Both + Key Hex	Char
&1	26	&	26	&	31	1	31	1
"2	22	"	22	"	32	2	32	2
'3	27	'	27	'	33	3	33	3
(4	28	(	28	(	34	4	34	4
ç5	5C	ç	1C	FS	35	5	1C	FS
è6	7D	è	7D	è	36	6	36	6
)7	29	)	29	)	37	7	37	7
£8	23	£	23	£	38	8	38	8
à9	7B	à	7B	à	39	9	39	9
ó0	5D	ó	1D	GS	30	0	1D	GS
í¹	7E	í	1E	RS	5E	¹	1E	RS
\$²	24	\$	24	\$	2A	²	2A	²
ú³	60	ú	60	ú	25	³	25	³
§⁴	40	§	00	NUL	5B	⁴	1B	ESC
<⁵	3C	<	3C	<	3E	>	3E	>
,⁶	2C	,	2C	,	3F	?	3F	?
:⁷	3B	:	3B	:	2E	.	2E	.
/⁸	3A	/	3A	/	2F	/	2F	/
ò⁹	7C	ò	7C	ò	21	!	21	!

### G.1.8 Western Spanish Keyboard

With the keyboard switch up, the Western (that is, American) Spanish model of the IIC keyboard layout is as shown in Figure G-1, and keystrokes produce the ASCII codes shown in Table G-1.

With the keyboard switch down, the Western Spanish model keyboard layout is as shown in Figure G-9. The change in ASCII code production (from that in Table G-1) is shown in Table G-6.

Figure G-9. Western Spanish Keyboard (Keyboard Switch Down)

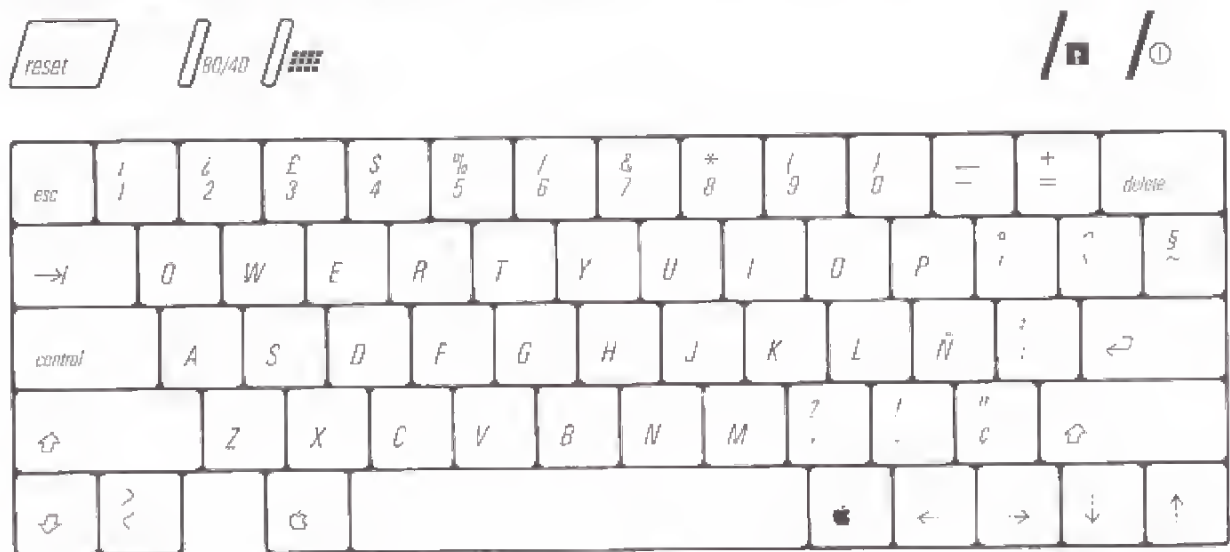


Table G-6. Western Spanish Keyboard Code Differences From Table G-1

Key	Key Alone Hex	Char	CONTROL + Key Hex	Char	SHIFT + Key Hex	Char	Both + Key Hex	Char
2"	32	2	32	2	22	"	22	"
3£	33	3	33	3	23	£	23	£
6&	36	6	00	NUL	26	&	00	NUL
7/	37	7	37	7	2F	/	2F	/
8(	38	8	38	8	28	(	28	(
9)	39	9	39	9	29	)	29	)
0=	30	0	30	0	3D	=	3D	=
?'	27	'	27	'	3F	?	3F	?
¿	60	¿	60	¿	5D	¿	5D	¿
~	7E	~	1E	RS	5E	~	1E	RS
+*	2B	*	1B	ESC	2A	+	1B	ESC
Ñ	7C	Ñ	1C	FS	5C	Ñ	1C	FS
¢	7D	¢	7D	¢	5B	¢	5B	¢
°\$	7B	\$	00	NUL	40	°	00	NUL
<>	3C	<	1E	RS	3E	>	1E	RS
..	2C	..	2C	..	3B	..	3B	..
..	2E	..	2E	..	3A	..	3A	..

## **G.2 ASCII Character Sets**

Table G-7 lists the ASCII (American National Standard Code for Information Interchange) codes that the Apple IIc uses, as well as the decimal and hexadecimal equivalents. Where there are differences between character sets, a circled number in the main table refers to a column in the lower part of the table.

Table G-7. ASCII Code Equivalents

ASCII	DEC	HEX	ASCII	DEC	HEX	ASCII	DEC	HEX	ASCII	DEC	HEX
NUL	00	00	SP	32	20	*②	64	40	*⑦	96	60
SOH	01	01	!	33	21	A	65	41	a	97	61
STX	02	02	"	34	22	B	66	42	b	98	62
ETX	03	03	*①	35	23	C	67	43	c	99	63
EOT	04	04	-①	36	24	D	68	44	d	100	64
ENO	05	05	%	37	25	E	69	45	e	101	65
ACK	06	06	&	38	26	F	70	46	f	102	66
BEL	07	07	'	39	27	G	71	47	g	103	67
BS	08	08	{	40	28	H	72	48	h	104	68
HT	09	09	}	41	29	I	73	49	i	105	69
LF	10	0A	*	42	2A	J	74	4A	j	106	6A
VT	11	0B	+	43	2B	K	75	4B	k	107	6B
FF	12	0C	.	44	2C	L	76	4C	l	108	6C
CR	13	0D	-	45	2D	M	77	4D	m	109	6D
SO	14	0E	-	46	2E	N	78	4E	n	110	6E
SI	15	0F	/	47	2F	O	79	4F	o	111	6F
DLE	16	10	0	48	30	P	80	50	p	112	70
DC1	17	11	1	49	31	Q	81	51	q	113	71
DC2	18	12	2	50	32	R	82	52	r	114	72
DC3	19	13	3	51	33	S	83	53	s	115	73
DC4	20	14	4	52	34	T	84	54	t	116	74
NAK	21	15	5	53	35	U	85	55	u	117	75
SYN	22	16	6	54	36	V	86	56	v	118	76
ETB	23	17	7	55	37	W	87	57	w	119	77
CAN	24	18	8	56	38	X	88	58	x	120	78
EM	25	19	9	57	39	Y	89	59	y	121	79
SUB	26	1A	:	58	3A	Z	90	5A	z	122	7A
ESC	27	1B	:	59	3B	*③	91	5B	*⑧	123	7B
FS	28	1C	<	60	3C	*④	92	5C	*⑨	124	7C
GS	29	1D	=	61	3D	*⑤	93	5D	*⑩	125	7D
RS	30	1E	>	62	3E	*⑥	94	5E	*⑪	126	7E
US	31	1F	?	63	3F	—	95	5F	DEL	127	7F

	①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩	⑪	
Hexadecimal	23	24	40	5B	5C	5D	5E	60	7B	7C	7D	7E
English (USA)	#	\$	@	[	\	]	^	_	{		}	~
English (UK)	£	\$	@	[	\	]	^	_	{		}	~
Deutsch	#	\$	§	À	Ö	Ü	·	·	ä	ö	ü	ß
Français	£	\$	à	°	ç	§	·	·	é	ù	è	
Italiano	£	\$	§	°	ç	é	·	ù	à	ò	è	ì
Españolo	£	\$	§		Ñ	¿	·	·	°	ñ	ç	·

## **G.3 Certifications**

In the countries where they are applicable, these certifications replace the USA FCC Class B notice printed on the inside front cover of this manual. The safety instructions apply to all countries.

---

### **G.3.1 Radio Interference**

This product is designed to comply with specification VDE 0871/6.78, Radio Frequency Interference Suppression of Radio Frequency Equipment, Level B.

---

### **G.3.2 Product Safety**

This product is designed to meet the requirements of safety standard IEC 380, Safety of Electrically Energized Office Machines.

---

### **G.3.3 Important Safety Instructions**

This equipment is intended to be electrically grounded. This product is equipped with a plug having a third (grounding) pin. This plug will fit only into a grounding-type alternating current outlet. This is a safety feature.

If you are unable to insert the plug into the outlet, contact a licensed electrician to replace the outlet and, if necessary, install a grounding conductor.

Do not defeat the purpose of the grounding-type plug.

## **G.4 Power Supply Specifications**

The basic specifications of the power supply furnished with the Apple IIc for use in Europe and other countries having 50 Hz alternating current are shown in Table G-8.

*Table G-8. 50 Hz Power Supply Specifications*

<b>Line voltage</b>	199 to 255 VAC, 50 Hz
<b>Maximum input power consumption</b>	25 W
<b>Supply voltage</b>	+15 V DC (nominal)
<b>Supply current</b>	1.2 A (nominal)





# *Conversion Tables*

This appendix briefly discusses bits and bytes and what they can represent. It also contains conversion tables for hexadecimal to decimal and negative decimal, for low-resolution display dot patterns, display color values, and a number of 8-bit codes.

These tables are intended for convenient reference. This appendix is not intended as a tutorial for the materials discussed. The brief section introductions are for orientation only.

## ***H.1 Bits and Bytes***

This section discusses the relationships between bit values and their position within a byte. The following are some rules of thumb regarding the 65C02.

- A bit is a binary digit; it can be either a 0 or a 1.
- A bit can be used to represent any two-way choice. Some choices that a bit can represent in the Apple IIc are listed in Table H-1.

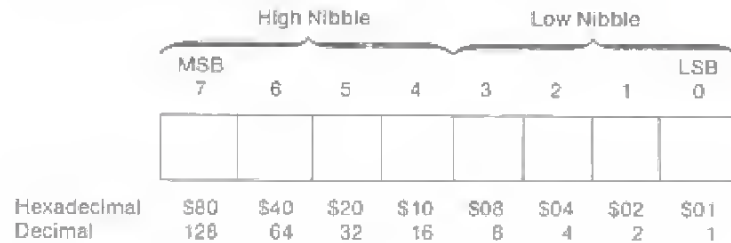
**Table H-1. What a Bit Can Represent**

Context	Representing	0 =	1 =
Binary number	Place value	0	1 x that power of 2
Logic	Condition	False	True
Any switch	Position	Off	On
Any switch	Position	Clear†	Set
Serial transfer	Beginning	Start	Carrier (no information yet)
Serial transfer	Data	0 value	1 value
Serial transfer	Parity	SPACE	MARK
Serial transfer	End		Stop bit(s)
Serial transfer	Communication state	BREAK	Carrier
P reg. bit N	Neg. result?	No	Yes
P reg. bit V	Overflow?	No	Yes
P reg. bit B	BRK command?	No	Yes
P reg. bit D	Decimal mode?	No	Yes
P reg. bit I	IRQ interrupts	Enabled	Disabled (masked out)
P reg. bit Z	Zero result?	No	Yes
P reg. bit C	Carry required?	No	Yes

† Sometimes ambiguously termed *reset*.

- Bits can also be combined in groups of any size to represent numbers. Most of the commonly used sizes are multiples of four bits.
- Four bits comprise a nibble (sometimes spelled *nybble*).
- One nibble can represent any of 16 values. Each of these values is assigned a number from 0 through 9 and (because our decimal system has only ten of the sixteen digits we need) A through F.
- Eight bits (two nibbles) make a byte (Figure H-1).

Figure H-1. Bits, Nibbles, and Bytes



Binary	Hexadecimal	Decimal
0000	\$0	0
0001	\$1	1
0010	\$2	2
0011	\$3	3
0100	\$4	4
0101	\$5	5
0110	\$6	6
0111	\$7	7
1000	\$8	8
1001	\$9	9
1010	\$A	10
1011	\$B	11
1100	\$C	12
1101	\$D	13
1110	\$E	14
1111	\$F	15

- One byte can represent any of 16 x 16 or 256 values. The value can be specified by exactly two hexadecimal digits.
- Bits within a byte are numbered from bit 0 on the right to bit 7 on the left.
- The bit number is the same as the power of 2 that it represents, in a manner completely analogous to the digits in a decimal number.
- One memory position in the Apple IIc contains one eight-bit byte of data.
- How byte values are interpreted depends on whether the byte is an instruction in a language, part or all of an address, an ASCII code, or some other form of data. Tables H-6 through H-13 list some of the ways bytes are commonly interpreted.

- Two bytes make a word. The sixteen bits of a word can represent any one of 256 x 256 or 65536 different values.
- The 65C02 uses a 16-bit word to represent memory locations. It can therefore distinguish among 65536 (64K) locations at any given time.
- A memory location is one byte of a 256-byte page. The low-order byte of an address specifies this byte. The high-order byte specifies the memory page the byte is on.

## H.2 Hexadecimal and Decimal

Use Table H-2 for conversion of hexadecimal and decimal numbers.

*Table H-2. Hexadecimal/Decimal Conversion*

Digit	\$x000	\$0x00	\$00x0	\$000x
F	61440	3840	240	15
E	57344	3584	224	14
D	53248	3328	208	13
C	49152	3072	192	12
B	45056	2816	176	11
A	40960	2560	160	10
9	36864	2304	144	9
8	32768	2048	128	8
7	28672	1792	112	7
6	24576	1536	96	6
5	20480	1280	80	5
4	16384	1024	64	4
3	12288	768	48	3
2	8192	512	32	2
1	4096	256	16	1

To convert a hexadecimal number to a decimal number, find the decimal numbers corresponding to the positions of each hexadecimal digit. Write them down and add them up.

**Examples:**

$$\text{\$3C} = ?$$

$$\text{\$30} = 48$$

$$\text{\$0C} = 12$$

-----

$$\text{\$3C} = 60$$

$$\text{\$FD47} = ?$$

$$\text{\$F000} = 61440$$

$$\text{\$D00} = 3328$$

$$\text{\$40} = 64$$

$$\text{\$7} = 7$$

-----

$$\text{\$FD47} = 64839$$



To convert a decimal number to hexadecimal, subtract from the decimal number the largest decimal entry in the table that is less than it. Write down the hexadecimal digit (noting its place value) also. Now subtract the largest decimal number in the table that is less than the decimal remainder, and write down the next hexadecimal digit. Continue until you have zero left. Add up the hexadecimal numbers.

**Example:**

16215 = \$ ?

16215 - 12288 = 3927  
 3927 - 3840 = 87  
 87 - 80 = 7  
           7

12288 = \$7000  
 3840 = \$F00  
 80 = \$50  
 7 = \$7

-----  
 16215 = \$7F57

### **H.3 Hexadecimal and Negative Decimal**

If a number is larger than decimal 32767, Applesoft BASIC allows and Integer BASIC requires that you use the negative-decimal equivalent of the number. Table H-3 is set up to make it easy for you to convert a hexadecimal number directly to a negative decimal number.

**Table H-3. Decimal to Negative Decimal Conversion**

Digit	\$x000	\$\$0x00	\$\$00x0	\$\$\$000x
F	0	0	0	-1
E	-4096	-256	-16	-2
D	-8192	-512	-32	-3
C	-12288	-768	-48	-4
B	-16384	-1024	-64	-5
A	-20480	-1280	-80	-6
9	-24576	-1536	-96	-7
8	-28672	-1792	-112	-8
7		-2048	-128	-9
6		-2304	-144	-10
5		-2560	-160	-11
4		-2816	-176	-12
3		-3072	-192	-13
2		-3328	-208	-14
1		-3584	-224	-15
0		-3840	-240	-16

To perform this conversion, write down the four decimal numbers corresponding to the four hexadecimal digits (zeros included). Then add their values (ignoring their signs for a moment). The resulting number, with a minus sign in front of it, is the desired negative decimal number.

**Example:**

\$C010 = - ?

\$C000: - 12288

\$ 000: - 3840

\$ 10: - 224

\$ 0: - 16

-----

\$C010 - 16368

To convert a negative-decimal number directly to a positive decimal number, add it to 65536. (This addition ends up looking like subtraction.)

**Example:**

-151 = + ?

65536 + (-151) = 65536 - 151 = 65385

To convert a negative-decimal number to a hexadecimal number, first convert it to a positive decimal number, then use Table H-2.

## H.4 Graphics Bits and Pieces

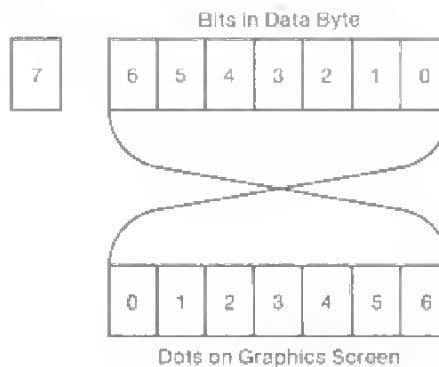
Table H-4 is a quick guide to the hexadecimal values corresponding to 7-bit high-resolution patterns on the display screen. Since the bits are displayed in reverse order, it takes some calculation to determine these values. Table H-4 should make it easy.

The *x* represents bit 7. Zeros represent bits that are off; ones represent bits that are on. Use the first hexadecimal value if bit 7 is to be off, and the second if it is to be on.

For example, to get bit pattern 00101110, use \$3A; for 10101110, use \$BA.

**Table H-4. Hexadecimal Values for High-Resolution Dot Patterns**

Bit pattern	( <i>x</i> = 0)	( <i>x</i> = 1)
x0000000	\$00	\$80
x0000001	\$40	\$C0
x0000010	\$20	\$A0
x0000011	\$60	\$E0
x0000100	\$10	\$90
x0000101	\$50	\$D0
x0000110	\$30	\$B0
x0000111	\$70	\$F0
x0001000	\$08	\$88
x0001001	\$48	\$C8
x0001010	\$28	\$A8
x0001011	\$68	\$E8
x0001100	\$18	\$98
x0001101	\$58	\$D8
x0001110	\$38	\$B8
x0001111	\$78	\$F8
x0010000	\$04	\$84
x0010001	\$44	\$C4
x0010010	\$24	\$A4
x0010011	\$64	\$E4
x0010100	\$14	\$94
x0010101	\$54	\$D4
x0010110	\$34	\$B4
x0010111	\$74	\$F4
x0011000	\$0C	\$8C
x0011001	\$4C	\$CC
x0011010	\$2C	\$AC
x0011011	\$6C	\$EC
x0011100	\$1C	\$9C
x0011101	\$5C	\$DC
x0011110	\$3C	\$BC
x0011111	\$7C	\$FC



**Table H-4—Continued. Hexadecimal Values for High-Resolution Dot Patterns**

Bit pattern	(x = 0)	(x = 1)
x0100000	\$02	\$82
x0100001	\$42	\$C2
x0100010	\$22	\$A2
x0100011	\$62	\$E2
x0100100	\$12	\$92
x0100101	\$52	\$D2
x0100110	\$32	\$B2
x0100111	\$72	\$F2
x0101000	\$0A	\$8A
x0101001	\$4A	\$CA
x0101010	\$2A	\$AA
x0101011	\$6A	\$EA
x0101100	\$1A	\$9A
x0101101	\$5A	\$DA
x0101110	\$3A	\$BA
x0101111	\$7A	\$FA
x0110000	\$06	\$86
x0110001	\$46	\$C6
x0110010	\$26	\$A6
x0110011	\$66	\$E6
x0110100	\$16	\$96
x0110101	\$56	\$D6
x0110110	\$36	\$B6
x0110111	\$76	\$F6
x0111000	\$0E	\$8E
x0111001	\$4E	\$CE
x0111010	\$2E	\$AE
x0111011	\$6E	\$EE
x0111100	\$1E	\$9E
x0111101	\$5E	\$DE
x0111110	\$3E	\$BE
x0111111	\$7E	\$FE
x1000000	\$01	\$81
x1000001	\$41	\$C1
x1000010	\$21	\$A1
x1000011	\$61	\$E1
x1000100	\$11	\$91
x1000101	\$51	\$D1
x1000110	\$31	\$B1
x1000111	\$71	\$F1
x1001000	\$09	\$89
x1001001	\$49	\$C9
x1001010	\$29	\$A9
x1001011	\$69	\$E9
x1001100	\$19	\$99
x1001101	\$59	\$D9
x1001110	\$39	\$B9
x1001111	\$79	\$F9

**Table H-4—Continued.** Hexadecimal Values for High-Resolution Dot Patterns

Bit pattern	(x=0)	(x=1)
x1010000	\$05	\$85
x1010001	\$45	\$C5
x1010010	\$25	\$A5
x1010011	\$65	\$E5
x1010100	\$15	\$95
x1010101	\$55	\$D5
x1010110	\$35	\$B5
x1010111	\$75	\$F5
x1011000	\$0D	\$8D
x1011001	\$4D	\$CD
x1011010	\$2D	\$AD
x1011011	\$6D	\$ED
x1011100	\$1D	\$9D
x1011101	\$5D	\$DD
x1011110	\$3D	\$BD
x1011111	\$7D	\$FD
x1100000	\$03	\$83
x1100001	\$43	\$C3
x1100010	\$23	\$A3
x1100011	\$63	\$E3
x1100100	\$13	\$93
x1100101	\$53	\$D3
x1100110	\$33	\$B3
x1100111	\$73	\$F3
x1101000	\$0B	\$8B
x1101001	\$4B	\$CB
x1101010	\$2B	\$AB
x1101011	\$6B	\$EB
x1101100	\$1B	\$9B
x1101101	\$5B	\$DB
x1101110	\$3B	\$BB
x1101111	\$7B	\$FB
x1110000	\$07	\$87
x1110001	\$47	\$C7
x1110010	\$27	\$A7
x1110011	\$67	\$E7
x1110100	\$17	\$97
x1110101	\$57	\$D7
x1110110	\$37	\$B7
x1110111	\$77	\$F7
x1111000	\$0F	\$8F
x1111001	\$4F	\$CF
x1111010	\$2F	\$AF
x1111011	\$6F	\$EF
x1111100	\$1F	\$9F
x1111101	\$5F	\$DF
x1111110	\$3F	\$BF
x1111111	\$7F	\$FF

---

## H.5 Peripheral Identification Numbers

Many Apple products now use Peripheral Identification Numbers (called **PIN numbers**) as shorthand for serial device characteristics. The Apple II Series *System Utilities Disk* presents a menu from which to select the characteristics of, say, a printer or modem. From the selections made, it generates a PIN for the user. Other products have a ready-made PIN that the user can simply type in.

Table H-5 is a definition of the PIN number digits. When communication mode is selected, the seventh digit is ignored.

**Example:**

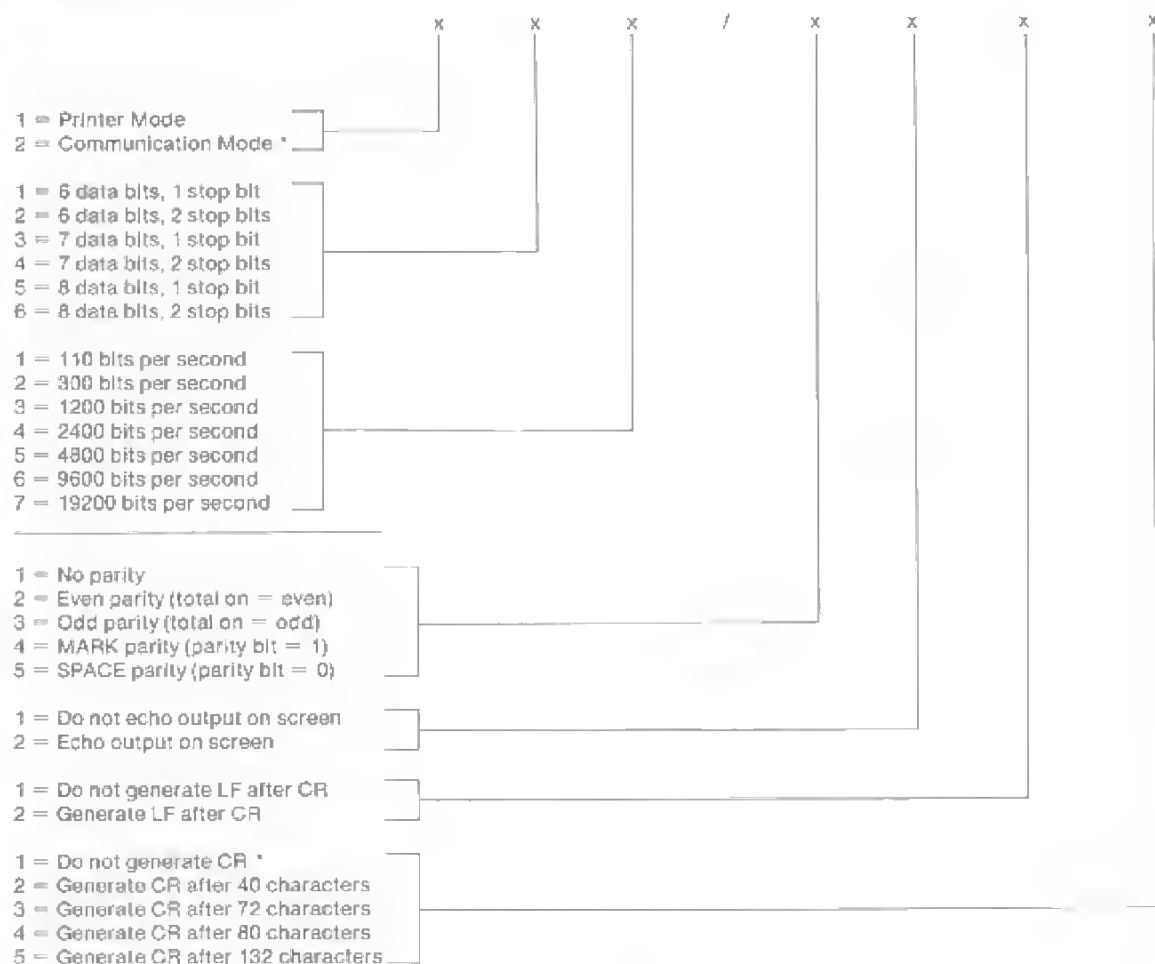
252/1111 means:

Communication mode  
8 data bits, 1 stop bit  
300 baud (bits per second)

No parity  
Do not echo output to display  
No linefeed after carriage return  
Do not generate carriage returns



Table H-5. PIN Numbers



\* If you select Communication Mode, then seventh digit must equal 1.  
This value is supplied automatically when you use the UUD.

## H.6 Eight-Bit Code Conversions

Tables H-6 through H-13 show the entire ASCII character set twice: once with the high bit off, and once with it on. Here is how to interpret these tables.

- The **Binary** column has the 8-bit code for each ASCII character.
- The first 128 ASCII entries represent 7-bit ASCII codes plus a high-order bit of 0 (SPACE parity or Pascal)—for example, 01001000 for the letter *H*.
- The last 128 ASCII entries (from 128 through 255) represent 7-bit ASCII codes plus a high-order bit of 1 (MARK parity or BASIC)—for example, 11001000 for the letter *H*.
- A transmitted or received ASCII character will take whichever form (in the communication register) is appropriate if odd or even parity is selected—for example, 11001000 for an odd-parity H, 01001000 for an even-parity H.
- The **ASCII Char** column gives the ASCII character name.
- The **Interpretation** column spells out the meaning of special symbols and abbreviations, where necessary.
- The **What to Type** column indicates what keystrokes generate the ASCII character (where it is not obvious). The numbers between columns refer to footnotes.
- The columns marked **Pri** and **Alt** indicate what displayed character results from each code when using the primary or alternate display character set, respectively. Boldface is used for inverse characters; italic is used for flashing characters.

Note that the values \$40 through \$5F (and \$C0 through \$DF) in the alternate character set are displayed as MouseText characters (Figure 5-1) if the firmware is set to do so (section 5.2.2), or if the firmware is bypassed.

**Note:** The primary and alternate displayed character sets in Tables H-6 through H-13 are the result of firmware mapping. The CHARGEN ROM actually contains only one character set. The firmware mapping procedure is described in section 3.36.

*Table H-6. Control Characters, High Bit Off*

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
0000000	0	\$00	NUL	Blank (null)	(CONTROL)-(0)	@	@
0000001	1	\$01	SOH	Start of Header	(CONTROL)-(A)	A	A
0000010	2	\$02	STX	Start of Text	(CONTROL)-(B)	B	B
0000011	3	\$03	ETX	End of Text	(CONTROL)-(C)	C	C
0000100	4	\$04	EOT	End of Transm.	(CONTROL)-(D)	D	D
0000101	5	\$05	ENQ	Enquiry	(CONTROL)-(E)	E	E
0000110	6	\$06	ACK	Acknowledge	(CONTROL)-(F)	F	F
0000111	7	\$07	BEL	Bell	(CONTROL)-(G)	G	G
0001000	8	\$08	BS	Backspace	(CONTROL)-(H)	H	H
					or (-)		
0001001	9	\$09	HT	Horizontal Tab	(CONTROL)-(I)	I	I
					or (TAB)		
0001010	10	\$0A	LF	Line Feed	(CONTROL)-(J)	J	J
					or (L)		
0001011	11	\$0B	VT	Vertical Tab	(CONTROL)-(K)	K	K
					or (V)		
0001100	12	\$0C	FF	Form Feed	(CONTROL)-(L)	L	L
0001101	13	\$0D	CR	Carriage Return	(CONTROL)-(M)	M	M
					or (RETURN)		
0001110	14	\$0E	SO	Shift Out	(CONTROL)-(N)	N	N
0001111	15	\$0F	SI	Shift In	(CONTROL)-(O)	O	O
0010000	16	\$10	DLE	Data Link Escape	(CONTROL)-(P)	P	P
0010001	17	\$11	DC1	Device Control 1	(CONTROL)-(Q)	Q	Q
0010010	18	\$12	DC2	Device Control 2	(CONTROL)-(R)	R	R
0010011	19	\$13	DC3	Device Control 3	(CONTROL)-(S)	S	S
0010100	20	\$14	DC4	Device Control 4	(CONTROL)-(T)	T	T
0010101	21	\$15	NAK	Neg. Acknowledge	(CONTROL)-(U)	U	U
					or (-)		
0010110	22	\$16	SYN	Synchronization	(CONTROL)-(V)	V	V
0010111	23	\$17	ETB	End of Text Blk.	(CONTROL)-(W)	W	W
0011000	24	\$18	CAN	Cancel	(CONTROL)-(X)	X	X
0011001	25	\$19	EM	End of Medium	(CONTROL)-(Y)	Y	Y
0011010	26	\$1A	SUB	Substitute	(CONTROL)-(Z)	Z	Z
0011011	27	\$1B	ESC	Escape	(CONTROL)-(I)	[	[
					or (ESC)		
0011100	28	\$1C	FS	File Separator	(CONTROL)-(I)	\	\
0011101	29	\$1D	GS	Group Separator	(CONTROL)-(I)	] ^	] ^
0011110	30	\$1E	RS	Record Separator	(CONTROL)-(I)	^	^
0011111	31	\$1F	US	Unit Separator	(CONTROL)-(I)	-	-

Table H-7. Special Characters, High Bit Off

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	All
0100000	32	\$20	SP	Space	SPACE bar		
0100001	33	\$21				!	!
0100010	34	\$22	"			"	"
0100011	35	\$23	#			#	#
0100100	36	\$24	\$			\$	\$
0100101	37	\$25	%			%	%
0100110	38	\$26	&			&	&
0100111	39	\$27	'	Closing Quote		'	'
0101000	40	\$28	(			(	(
0101001	41	\$29	)			)	)
0101010	42	\$2A	*			*	*
0101011	43	\$2B	+			+	+
0101100	44	\$2C	,	Comma		,	,
0101101	45	\$2D	-	Hyphen		-	-
0101110	46	\$2E	.	Period		.	.
0101111	47	\$2F	/			/	/
0110000	48	\$30	0			0	0
0110001	49	\$31	1			1	1
0110010	50	\$32	2			2	2
0110011	51	\$33	3			3	3
0110100	52	\$34	4			4	4
0110101	53	\$35	5			5	5
0110110	54	\$36	6			6	6
0110111	55	\$37	7			7	7
0111000	56	\$38	8			8	8
0111001	57	\$39	9			9	9
0111010	58	\$3A	:			:	:
0111011	59	\$3B	;			;	;
0111100	60	\$3C	<			<	<
0111101	61	\$3D	=			=	=
0111110	62	\$3E	>			>	>
0111111	63	\$3F	?			?	?

Table H-8. Uppercase Characters, High Bit Off

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pr	Alt
1000000	64	\$40	@			@	ⓐ
1000001	65	\$41	A			A	ⓐ
1000010	66	\$42	B			B	ⓑ
1000011	67	\$43	C			C	ⓒ
1000100	68	\$44	D			D	ⓓ
1000101	69	\$45	E			E	ⓔ
1000110	70	\$46	F			F	ⓕ
1000111	71	\$47	G			G	ⓖ
1001000	72	\$48	H			H	ⓓ
1001001	73	\$49	I			I	ⓔ
1001010	74	\$4A	J			J	ⓕ
1001011	75	\$4B	K			K	ⓖ
1001100	76	\$4C	L			L	ⓓ
1001101	77	\$4D	M			M	ⓔ
1001110	78	\$4E	N			N	ⓕ
1001111	79	\$4F	O			O	ⓖ
1010000	80	\$50	P			P	ⓓ
1010001	81	\$51	Q			Q	ⓔ
1010010	82	\$52	R			R	ⓕ
1010011	83	\$53	S			S	ⓖ
1010100	84	\$54	T			T	ⓓ
1010101	85	\$55	U			U	ⓔ
1010110	86	\$56	V			V	ⓕ
1010111	87	\$57	W			W	ⓖ
1011000	88	\$58	X			X	ⓓ
1011001	89	\$59	Y			Y	ⓔ
1011010	90	\$5A	Z			Z	ⓕ
1011011	91	\$5B		Opening Bracket		[	ⓖ
1011100	92	\$5C	.	Reverse Slant		/	ⓓ
1011101	93	\$5D		Closing Bracket		]	ⓔ
1011110	94	\$5E	^	Caret		^	ⓕ
1011111	95	\$5F	_	Underline		_	ⓖ

**Table H-9. Lowercase Characters, High Bit Off**

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
1100000	96	\$60	`	Opening Quote		!	~
1100001	97	\$61	a			"	a
1100010	98	\$62	b			#	b
1100011	99	\$63	c			\$	c
1100100	100	\$64	d			%	d
1100101	101	\$65	e			&	e
1100110	102	\$66	f			'	f
1100111	103	\$67	g			(	g
1101000	104	\$68	h			)	h
1101001	105	\$69	i			*	i
1101010	106	\$6A	j			+	j
1101011	107	\$6B	k			,	k
1101100	108	\$6C	l			-	l
1101101	109	\$6D	m			.	m
1101110	110	\$6E	n			/	n
1101111	111	\$6F	o			0	o
1110000	112	\$70	p			1	p
1110001	113	\$71	q			2	q
1110010	114	\$72	r			3	r
1110011	115	\$73	s			4	s
1110100	116	\$74	t			5	t
1110101	117	\$75	u			6	u
1110110	118	\$76	v			7	v
1110111	119	\$77	w			8	w
1111000	120	\$78	x			9	x
1111001	121	\$79	y			:	y
1111010	122	\$7A	z			; < =	z
1111011	123	\$7B	{	Opening Brace		> ?	[
1111100	124	\$7C		Vertical Line			
1111101	125	\$7D	}	Closing Brace			]
1111110	126	\$7E	~	Overline (Tilde)			~
1111111	127	\$7F	DEL	Delete/Rubout			DEL



Table H-10. Control Characters, High Bit On

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
10000000	128	\$80	NUL	Blank (null)	CONTROL-@	@	@
10000001	129	\$81	SOH	Start of Header	CONTROL-A	A	A
10000010	130	\$82	STX	Start of Text	CONTROL-B	B	B
10000011	131	\$83	ETX	End of Text	CONTROL-C	C	C
10000100	132	\$84	EOT	End of Transm.	CONTROL-D	D	D
10000101	133	\$85	ENQ	Enquiry	CONTROL-E	E	E
10000110	134	\$86	ACK	Acknowledge	CONTROL-F	F	F
10000111	135	\$87	BEL	Bell	CONTROL-G	G	G
10001000	136	\$88	BS	Backspace	CONTROL-H	H	H
10001001	137	\$89	HT	Horizontal Tab	or CONTROL-I or TAB	I	I
10001010	138	\$8A	LF	Line Feed	CONTROL-J	J	J
10001011	139	\$8B	VT	Vertical Tab	or CONTROL-K or I	K	K
10001100	140	\$8C	FF	Form Feed	CONTROL-L	L	L
10001101	141	\$8D	CR	Carriage Return	CONTROL-M or RETURN	M	M
10001110	142	\$8E	SO	Shift Out	CONTROL-N	N	N
10001111	143	\$8F	SI	Shift In	CONTROL-O	O	O
10010000	144	\$90	DLE	Data Link Escape	CONTROL-P	P	P
10010001	145	\$91	DC1	Device Control 1	CONTROL-Q	Q	Q
10010010	146	\$92	DC2	Device Control 2	CONTROL-R	R	R
10010011	147	\$93	DC3	Device Control 3	CONTROL-S	S	S
10010100	148	\$94	DC4	Device Control 4	CONTROL-T	T	T
10010101	149	\$95	NAK	Neg. Acknowledge	CONTROL-U	U	U
10010110	150	\$96	SYN	Synchronization	or CONTROL-V	V	V
10010111	151	\$97	ETB	End of Text Blk.	CONTROL-W	W	W
10011000	152	\$98	CAN	Cancel	CONTROL-X	X	X
10011001	153	\$99	EM	End of Medium	CONTROL-Y	Y	Y
10011010	154	\$9A	SUB	Substitute	CONTROL-Z	Z	Z
10011011	155	\$9B	ESC	Escape	CONTROL-[ or ESC	[	[
10011100	156	\$9C	FS	File Separator	CONTROL-\	\	\
10011101	157	\$9D	GS	Group Separator	CONTROL-]	]	]
10011110	158	\$9E	RS	Record Separator	CONTROL-^	^	^
10011111	159	\$9F	US	Unit Separator	CONTROL-`	`	`

Table H-11. Special Characters, High Bit On

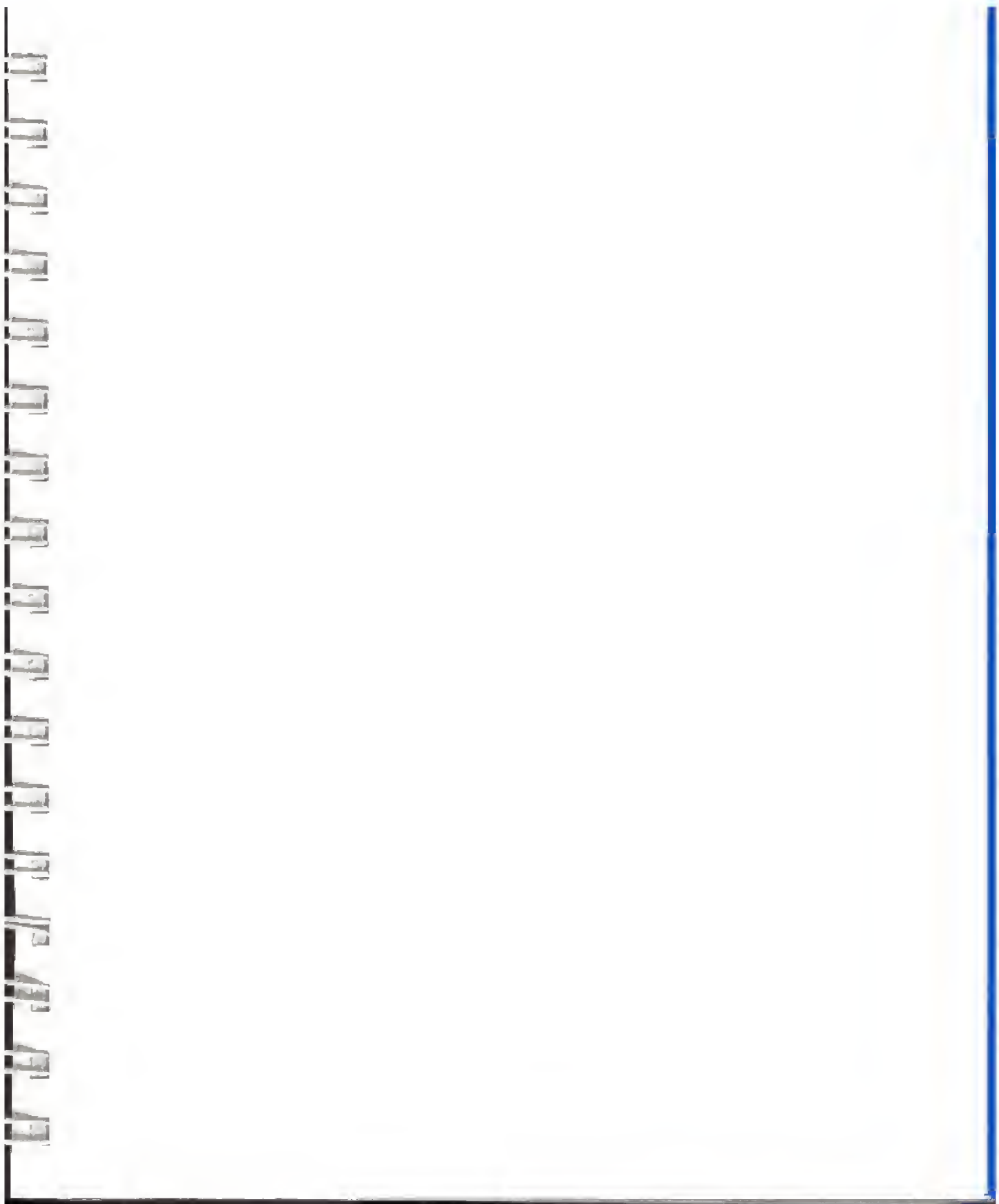
Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
10100000	160	\$A0	SP	Space	SPACE bar		
10100001	161	\$A1	!			!	!
10100010	162	\$A2	"			"	"
10100011	163	\$A3	#			#	#
10100100	164	\$A4	\$			\$	\$
10100101	165	\$A5	%			%	%
10100110	166	\$A6	&			&	&
10100111	167	\$A7	'	Closed Quote (acute accent)		'	'
10101000	168	\$A8	(			(	(
10101001	169	\$A9	)			)	)
10101010	170	\$AA	*			*	*
10101011	171	\$AB	+			+	+
10101100	172	\$AC	,	Comma		,	,
10101101	173	\$AD	-	Hyphen		-	-
10101110	174	\$AE	.	Period		.	.
10101111	175	\$AF	/			/	/
10110000	176	\$B0	0			0	0
10110001	177	\$B1	1			1	1
10110010	178	\$B2	2			2	2
10110011	179	\$B3	3			3	3
10110100	180	\$B4	4			4	4
10110101	181	\$B5	5			5	5
10110110	182	\$B6	6			6	6
10110111	183	\$B7	7			7	7
10111000	184	\$B8	8			8	8
10111001	185	\$B9	9			9	9
10111010	186	\$BA	:			:	:
10111011	187	\$BB	;			;	;
10111100	188	\$BC	<			<	<
10111101	189	\$BD	=			=	=
10111110	190	\$BE	>			>	>
10111111	191	\$BF	?			?	?

Table H-12. Uppercase Characters, High Bit On

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
11000000	192	\$C0	@			@	@
11000001	193	\$C1	A			A	A
11000010	194	\$C2	B			B	B
11000011	195	\$C3	C			C	C
11000100	196	\$C4	D			D	D
11000101	197	\$C5	E			E	E
11000110	198	\$C6	F			F	F
11000111	199	\$C7	G			G	G
11001000	200	\$C8	H			H	H
11001001	201	\$C9	I			I	I
11001010	202	\$CA	J			J	J
11001011	203	\$CB	K			K	K
11001100	204	\$CC	L			L	L
11001101	205	\$CD	M			M	M
11001110	206	\$CE	N			N	N
11001111	207	\$CF	O			O	O
11010000	208	\$D0	P			P	P
11010001	209	\$D1	Q			Q	Q
11010010	210	\$D2	R			R	R
11010011	211	\$D3	S			S	S
11010100	212	\$D4	T			T	T
11010101	213	\$D5	U			U	U
11010110	214	\$D6	V			V	V
11010111	215	\$D7	W			W	W
11011000	216	\$D8	X			X	X
11011001	217	\$D9	Y			Y	Y
11011010	218	\$DA	Z			Z	Z
11011011	219	\$DB	[	Opening Bracket		[	[
11011100	220	\$DC	\	Reverse Slant		\	\
11011101	221	\$DD	]	Closing Bracket		]	]
11011110	222	\$DE	^	Caret		^	^
11011111	223	\$DF	_	Underline		_	_

**Table H-13. Lowercase Characters, High Bit On**

Binary	Dec	Hex	ASCII Char	Interpretation	What to Type	Pri	Alt
11100000	224	\$E0	'	Open Quote		'	'
11100001	225	\$E1	a			a	a
11100010	226	\$E2	b			b	b
11100011	227	\$E3	c			c	c
11100100	228	\$E4	d			d	d
11100101	229	\$E5	e			e	e
11100110	230	\$E6	f			f	f
11100111	231	\$E7	g			g	g
11101000	232	\$E8	h			h	h
11101001	233	\$E9	i			i	i
11101010	234	\$EA	j			j	j
11101011	235	\$EB	k			k	k
11101100	236	\$EC	l			l	l
11101101	237	\$ED	m			m	m
11101110	238	\$EE	n			n	n
11101111	239	\$EF	o			o	o
11110000	240	\$F0	p			p	p
11110001	241	\$F1	q			q	q
11110010	242	\$F2	r			r	r
11110011	243	\$F3	s			s	s
11110100	244	\$F4	t			t	t
11110101	245	\$F5	u			u	u
11110110	246	\$F6	v			v	v
11110111	247	\$F7	w			w	w
11111000	248	\$F8	x			x	x
11111001	249	\$F9	y			y	y
11111010	250	\$FA	z			z	z
11111011	251	\$FB	{	Opening Brace		{	{
11111100	252	\$FC		Vertical Line			
11111101	253	\$FD	}	Closing Brace		}	}
11111110	254	\$FE	~	Overline (Tilde)		~	~
11111111	255	\$FF	DEL	Delete (Rubout)	DELETE	DEL	DEL



# *Firmware Listings*



Appendix I comprises a listing of the source code for the Monitor, enhanced video firmware, and input/output firmware contained in the Apple IIc.

```

C100:      2 *****
C100:      3 *
C100:      4 * Apple //e
C100:      5 * Video Firmware and
C100:      6 * Monitor ROM Source
C100:      7 *
C100:      8 * COPYRIGHT 1977-1983 BY
C100:      9 * APPLE COMPUTER, INC.
C100:     10 *
C100:     11 * ALL RIGHTS RESERVED
C100:     12 *
C100:     13 * S. WOZNIAK          1977
C100:     14 * A. BAUM           1977
C100:     15 * JOHN A           NOV 1978
C100:     16 * R. AURICCHIO     SEP 1981
C100:     17 * E. BEERNINK      1983
C100:     18 *
C100:     19 *****
C100:     20 *
C100:     21 * ZERO PAGE EQUATES
C100:     22 *
C100:    0000 23 LOCO      EQU   $00      ;vector for autostart from disk
C100:    0001 24 LOCI      EQU   $01
C100:    0020 25 WNDLFT    EQU   $20      ;left edge of text window
C100:    0021 26 WNDWDT    EQU   $21      ;width of text window
C100:    0022 27 WNDTOP    EQU   $22      ;top of text window
C100:    0023 28 WNDBTM    EQU   $23      ;bottom+1 of text window
C100:    0024 29 CH        EQU   $24      ;cursor horizontal position
C100:    0025 30 CV        EQU   $25      ;cursor vertical position
C100:    0026 31 GBASL     EQU   $26      ;lo-res graphics base addr.
C100:    0027 32 GBASH     EQU   $27
C100:    0028 33 BASL      EQU   $28      ;text base address
C100:    0029 34 BASH      EQU   $29
C100:    002A 35 BAS2L     EQU   $2A      ;temp base for scrolling
C100:    002B 36 BAS2H     EQU   $2B
C100:    002C 37 H2        EQU   $2C      ;temp for lo-res graphics
C100:    002C 38 LMNEM     EQU   $2C      ;temp for mnemonic decoding
C100:    002D 39 V2        EQU   $2D      ;temp for lo-res graphics
C100:    002D 40 RMNEM     EQU   $2D      ;temp for mnemonic decoding
C100:    002E 41 MASK      EQU   $2E      ;color mask for lo-res gr.
C100:    002E 42 FORMAT    EQU   $2E      ;temp for opcode decode
C100:    002F 43 LENGTH    EQU   $2F      ;temp for opcode decode
C100:    0030 44 COLOR     EQU   $30      ;color for lo-res graphics
C100:    0031 45 MODE      EQU   $31      ;Monitor mode
C100:    0032 46 INVFLG    EQU   $32      ;normal/inverse(/flash)
C100:    0033 47 PROMPT    EQU   $33      ;prompt character
C100:    0034 48 YSAV      EQU   $34      ;position in Monitor command
C100:    0035 49 YSAV1     EQU   $35      ;temp for Y register
C100:    0036 50 CSWL      EQU   $36      ;character output hook
C100:    0037 51 CSWH      EQU   $37
C100:    0038 52 KSWL      EQU   $38      ;character input hook
C100:    0039 53 KSWH      EQU   $39
C100:    003A 54 PCL       EQU   $3A      ;temp for program counter
C100:    003B 55 PCH       EQU   $3B
C100:    003C 56 A1L       EQU   $3C      ;Monitor temp
C100:    003D 57 A1H       EQU   $3D      ;Monitor temp
C100:    003E 58 A2L       EQU   $3E      ;Monitor temp
C100:    003F 59 A2H       EQU   $3F      ;Monitor temp

```

```

C100:      0040  60 A3L      EQU   $40           ;Monitor temp
C100:      0041  61 A3H      EQU   $41           ;Monitor temp
C100:      0042  62 A4L      EQU   $42           ;Monitor temp
C100:      0043  63 A4H      EQU   $43           ;Monitor temp
C100:      0044  64 A5L      EQU   $44           ;Monitor temp
C100:      0045  65 A5H      EQU   $45           ;Monitor temp
C100:      66 *
C100:      67 * Note: In Apple II, //e, both interrupts and BRK destroyed
C100:      68 * location $45. Now only BRK destroys $45 (ACC) and it
C100:      69 * also destroys $44 (MACSTAT).
C100:      70 *
C100:      0044  71 MACSTAT   EQU   $44           ;Machine state after BRK
C100:      0045  72 ACC      EQU   $45           ;Acc after BRK
C100:      73 *
C100:      0046  74 XREG      EQU   $46           ;X reg after break
C100:      0047  75 YREG      EQU   $47           ;Y reg after break
C100:      0048  76 STATUS   EQU   $48           ;P reg after break
C100:      0049  77 SPNT      EQU   $49           ;SP after break
C100:      004E  78 RNDL      EQU   $4E           ;random counter low
C100:      004F  79 RNDH      EQU   $4F           ;random counter high
C100:      80 *
C100:      81 * Value equates
C100:      82 *
C100:      0006  83 GOODF8     EQU   $06           ;value of //e, lolly ID byte
C100:      0095  84 PICK      EQU   $95           ;CONTROL-U character
C100:      009B  85 ESC       EQU   $9B           ;what ESC generates
C100:      86 *
C100:      87 * Characters read by GETLN are placed in
C100:      88 * IN, terminated by a carriage return.
C100:      89 *
C100:      0200  90 IN         EQU   $0200          ;input buffer for GETLN
C100:      91 *
C100:      92 * Page 3 vectors
C100:      93 *
C100:      03F0  94 BRKV       EQU   $03F0          ;vectors here after break
C100:      03F2  95 SOFTEV     EQU   $03F2          ;vector for warm start
C100:      03F4  96 PWREDUP    EQU   $03F4          ;THIS MUST = EOR #A5 OF SOFTEV+1
C100:      03F5  97 AMPERV     EQU   $03F5          ;APPLESOFT & EXIT VECTOR
C100:      03F8  98 USRADR      EQU   $03F8          ;APPLESOFT USR function vector
C100:      03FB  99 NMI        EQU   $03FB          ;NMI vector
C100:      03FE 100 IRQLOC     EQU   $03FE          ;Maskable interrupt vector
C100:      0400 101 LINE1      EQU   $0400          ;first line of text screen
C100:      07F8 102 MSL0T     EQU   $07F8          ;owner of $C8 space
C100:      103 *
C100:      104 * HARDWARE EQUATES
C100:      105 *
C100:      C000 106 IOADR      EQU   $C000          ;for IN#, PR# vector
C100:      C000 107 KRD        EQU   $C000          ;>127 if keystroke
C100:      C000 108 CLR80COL   EQU   $C000          ;disable 80 column store
C100:      C001 109 SET80COL   EQU   $C001          ;enable 80 column store
C100:      C002 110 RDMAINRAM  EQU   $C002          ;read from main 48K RAM
C100:      C003 111 RDCARDRAM  EQU   $C003          ;read from alt. 48K RAM
C100:      C004 112 WRMAINRAM  EQU   $C004          ;write to main 48K RAM
C100:      C005 113 WRCARDRAM  EQU   $C005          ;write to alt. 48K RAM
C100:      C008 114 SETSTDZP   EQU   $C008          ;use main zero page/stack
C100:      C009 115 SETALTZP   EQU   $C009          ;use alt. zero page/stack
C100:      C00C 116 CLR80VID   EQU   $C00C          ;disable 80 column hardware
C100:      C00D 117 SET80VID   EQU   $C00D          ;enable 80 column hardware

```

C100:	C00E	118	CLRALTCHAR	EQU	\$C00E	;normal LC, flashing UC
C100:	C00F	119	SETALTCHAR	EQU	\$C00F	;normal inverse, LC; no flash
C100:	C010	120	KBDSTR8	EQU	\$C010	;turn off key pressed flag
C100:	C011	121	RDLCRNK2	EQU	\$C011	; >127 if LC bank 2 is in
C100:	C012	122	RDLGRAM	EQU	\$C012	; >127 if LC RAM read enabled
C100:	C013	123	RDRAMRD	EQU	\$C013	; >127 if reading main 48K
C100:	C014	124	RDRAMWRT	EQU	\$C014	; >127 if writing main 48K
C100:	C016	125	RDALTZP	EQU	\$C016	; >127 if Alt ZP and LC switched in
C100:	C018	126	RDECOL	EQU	\$C018	; >127 if 80 column store
C100:	C019	127	RDVBLBAR	EQU	\$C019	; >127 if not VBL
C100:	C01A	128	RDTEXT	EQU	\$C01A	; >127 if text (not graphics)
C100:	C01B	129	RDMIX	EQU	\$C01B	; >127 if mixed mode on
C100:	C01C	130	RDPAGE2	EQU	\$C01C	; >127 if TXTPAGE2 switched in
C100:	C01D	131	RDHIRE	EQU	\$C01D	; >127 if HIRE is on
C100:	C01E	132	ALTCHARSET	EQU	\$C01E	; >127 if alternate char set in use
C100:	C01F	133	RDSOVID	EQU	\$C01F	; >127 if 80 column hardware in
C100:	C020	134	TAPEOUT	EQU	\$C020	;what is this??
C100:	C030	135	SPKR	EQU	\$C030	;clicks the speaker
C100:	C050	136	TXTCLE	EQU	\$C050	;switch in graphics (not text)
C100:	C051	137	TXTSET	EQU	\$C051	;switch in text (not graphics)
C100:	C052	138	MIXCLE	EQU	\$C052	;clear mixed-mode
C100:	C053	139	MIXSET	EQU	\$C053	;set mixed-mode (4 lines text)
C100:	C054	140	TXTPAGE1	EQU	\$C054	;switch in text page 1
C100:	C055	141	TXTPAGE2	EQU	\$C055	;switch in text page 2
C100:	C056	142	LORES	EQU	\$C056	;low-resolution graphics
C100:	C057	143	HIRE	EQU	\$C057	;high-resolution graphics
C100:	C058	144	CLRANO	EQU	\$C058	
C100:	C059	145	SETANO	EQU	\$C059	
C100:	C05A	146	CLRANI	EQU	\$C05A	
C100:	C05B	147	SETANI	EQU	\$C05B	
C100:	C05C	148	CLRAN2	EQU	\$C05C	
C100:	C05D	149	SETAN2	EQU	\$C05D	
C100:	C05E	150	CLRAN3	EQU	\$C05E	
C100:	C05F	151	SETAN3	EQU	\$C05F	
C100:	C060	152	RD40SW	EQU	\$C060	; >127 if 40/80 switch in 40 pos
C100:	C061	153	BUTNO	EQU	\$C061	;open apple key
C100:	C062	154	BUTN1	EQU	\$C062	;closed apple key
C100:	C064	155	PADDLD	EQU	\$C064	;read paddle 0
C100:	C070	156	PTRIG	EQU	\$C070	;trigger the paddles
C100:	C081	157	ROMIN	EQU	\$C081	;switch in \$D000-\$FFFF ROM
C100:	C083	158	LCBANK2	EQU	\$C083	;switch in LC bank 2
C100:	C08B	159	LCBANK1	EQU	\$C08B	;switch in LC bank 1
C100:	CFFF	160	CLRROM	EQU	\$CFFF	;switch out \$C8 ROMs
C100:	E000	161	BASIC	EQU	\$E000	;BASIC entry point
C100:	E003	162	BASIC2	EQU	\$E003	;BASIC warm entry point
C100:		163	*			
C100:	04FB	164	VMODE	EQU	\$4FB+3	;OPERATING MODE
C100:		165	*			
C100:		166	* BASIC VMODE BITS			
C100:		167	*			
C100:		168	* 1..... - BASIC active			
C100:		169	* 0..... - Pascal active			
C100:		170	* .0.....			
C100:		171	* .1.....			
C100:		172	* ..0..... - Print control characters			
C100:		173	* ..1..... - Don't print ctrl chars			
C100:		174	* ...0.....			
C100:		175	* ...1.....			

```

C100:      176 * ....0... - Print control characters
C100:      177 * ....1... - Don't print ctrl chars.
C100:      178 * .....0.. -
C100:      179 * .....1.. -
C100:      180 * .....0. -
C100:      181 * .....1. -
C100:      182 * .....0 - Print mouse characters
C100:      183 * .....1 - Don't print mouse characters
C100:      184 *
C100: 0040 185 M.40      EQU   $40
C100: 0020 186 M.CTL2   EQU   $20      ;Don't print controls
C100: 0008 187 M.CTL    EQU   $08      ;Don't print controls
C100: 0001 188 M.MOUSE   EQU   $01      ;Don't print mouse chars
C100:      189 *
C100:      190 * Pascal Mode Bits
C100:      191 *
C100:      192 * 1..... - BASIC active
C100:      193 * 0..... - Pascal active
C100:      194 * .0.....
C100:      195 * .1.....
C100:      196 * ..0..... -
C100:      197 * ..1..... -
C100:      198 * ...0.... - Cursor always on
C100:      199 * ...1.... - Cursor always off
C100:      200 * ....0... - GOTOXY n/a
C100:      201 * ....1... - GOTOXY in progress
C100:      202 * .....0.. - Normal Video
C100:      203 * .....1.. - Inverse Video
C100:      204 * .....0. -
C100:      205 * .....1. -
C100:      206 * .....0 - Print mouse chars
C100:      207 * .....1 - Don't print mouse chars
C100:      208 *
C100: 0080 209 M.PASCAL   EQU   $80      ;Pascal active
C100: 0010 210 M.CURSOR   EQU   $10      ;Don't print cursor
C100: 0008 211 M.GOXY     EQU   $08      ;GOTOXY IN PROGRESS
C100: 0004 212 M.VMODE    EQU   $04
C100:      213 *
C100: 0478 214 ROMSTATE   EQU   $478      ;temp store of ROM state
C100: 04F8 215 TEMP1      EQU   $4F8      ;used by CTLCHAR
C100: 0578 216 TEMP2      EQU   $578      ;used by scroll
C100: 05F8 217 TEMP3      EQU   $5F8      ;used by scroll
C100:      218 *
C100: 047B 219 OLDCH       EQU   $478+3    ;last value of CH
C100: 057B 220 OURCH       EQU   $578+3    ;80-COL CH
C100: 05FB 221 OURCV       EQU   $5F8+3    ;CURSOR VERTICAL
C100: 067B 222 VFACTV      EQU   $678+3    ;Bit7-video firmware inactive
C100: 06FB 223 XCOORD      EQU   $6F8+3    ;X-COORD (GOTOXY)
C100: 077B 224 NXTCUR      EQU   $778+3    ;next cursor to display
C100: 07FB 225 CURSOR      EQU   $7F8+3    ;the current cursor char
C100:      17      INCLUDE SERIAL      ;Equates for serial code

```

```

C100:      3 *****
C100:      4 *
C100:      5 * Apple Lolly communications driver
C100:      6 *
C100:      7 * By
C100:      8 * Rich Williams
C100:      9 * August 1983
C100:     10 * November 5 - j.r.huston
C100:     11 *
C100:     12 *****
C100:     13 *
C100:     14 * Command codes
C100:     15 *
C100:     16 * Default command char is ctrl-A (^A)
C100:     17 *
C100:     18 *      ^AnnB: Set baud rate to nn
C100:     19 *      ^AnnD: Set data format bits to nn
C100:     20 *      ^AI:  Enable video echo
C100:     21 *      ^AK:  Disable CRLF
C100:     22 *      ^AL:  Enable CRLF
C100:     23 *      ^AnnN: Disable video echo & set printer width
C100:     24 *      ^AnnP: Set parity bits to nn
C100:     25 *      ^AQ   Quit terminal mode
C100:     26 *      ^AR   Reset the ACIA, IN#0 PR#0
C100:     27 *      ^AS   Send a 233 ms break character
C100:     28 *      ^AT   Enter terminal mode
C100:     29 *      ^AZ:   Zap control commands
C100:     30 *      ^Ax:   Set command char to ^x
C100:     31 *      ^AnnCR:Set printer width (CR = carriage return)
C100:     32 *
C100:     33 *****
C100:      C100      34 serslot   equ   $C100
C100:      C200      35 comslot   equ   $C200
C100:      36          MSB       ON
C100:      00BF      37 cmdcur    equ   '?'           ;Cursor while on command mode
C100:      00DF      38 termcur    equ   ' '           ;Cursor while in terminal mode
C100:      39          MSB       OFF
C100:      0091      40 xon        equ   $91           ;XON character
C100:      03B8      41 sermode    equ   $3B8           ;D7=1 if in cmd; D6=1 if term 479 & 47A
C100:      0438      42 astat      equ   $438           ;Acia status from int 4F9 & 4FA
C100:      04B8      43 pwidth     equ   $4B8           ;Printer width 579 & 57A
C100:      0538      44 extint     equ   $538           ;extint & typhed enable 5F9 & 5FA
C100:      05F9      45 extint2    equ   $5F9
C100:      05FA      46 typhed     equ   $5FA
C100:      0679      47 oldcur     equ   $679           ;Saves cursor while in command
C100:      067A      48 oldcur2    equ   $67A           ;Saves cursor while in terminal mode
C100:      0638      49 eschar     equ   $638           ;Current escape character 6F9 & 6FA
C100:      06B8      50 flags      equ   $6B8           ;D7 = Video echo D6 = CRLF 779 & 77A
C100:      0738      51 col        equ   $738           ;Current printer column 7F9 & 7FA
C100:      047F      52 number     equ   $47F           ;Number accumulated in command
C100:      04FF      53 aciabuf    equ   $4FF           ;Owner of serial buffer
C100:      057F      54 twser      equ   $57F           ;Storage pointer for serial buffer
C100:      05FF      55 twkey      equ   $5FF           ;Storage pointer for type ahead buffer
C100:      067F      56 trser      equ   $67F           ;Retrieve pointer for serial buffer
C100:      06FF      57 trkey      equ   $6FF           ;Retrieve buffer for type ahead buffer
C100:      0800      58 thbuf      equ   $800           ;Buffer in alt ram space
C100:      06F8      59 temp       equ   $6F8           ;Temp storage
C100:      BFF8      60 sdata      equ   $BFF8          ;+$N0+$90 is output port

```



```

C100:      BFF9    61 sstat      equ    $BFF9      ;ACIA status register
C100:      BFFA    62 scomd      equ    $BFFA      ;ACIA command register
C100:      BFFB    63 sentl      equ    $BFFB      ;ACIA control register
C100:      FF58    64 iorts      equ    $FF58      ;RTS opcode
C100:      18                                INCLUDE SER ;Printer port @ $C100

```



```

C100:          3 *org serslot
C100:2C 58 FF  4          bit   iorte      ;Set V to indicate initial entry
C103:70 0C    C111  5          bvs   entrl    ;Always taken
C105:38        6          sec          ;Input entry point
C106:90        7          dfb   $90         ;BCC opcode
C107:18        8          clc          ;
C108:88        9          clv          ;V = 0 since not initial entry
C109:50 06    C111 10          bvc   entrl    ;Always taken

C10B:01        12         dfb   $01         ;pascal signiture byte
C10C:31        13         dfb   $31         ;device signiture
C10D:E4        14         dfb   >plinit
C10E:EE        15         dfb   >plread
C10F:F6        16         dfb   >plwrite
C110:FB        17         dfb   >plstatus

C111:DA        19 entrl   phx          ;Save the reg
C112:A2 C1     20         ldx   #<serslot ;X = Cn
C114:4C 33 C2  21         jmp   setup      ;Set mslot, etc
C117:90 05    C11E  22 serport bcc   serisout ;Only output allowed
C119:20 4D CE  23         jar   zzquit     ;Reset the hooks
C11C:80 6A    C188  24         bra   done
C11E:0A        25 serisout asl   A         ;A = flags
C11F:7A        26         ply          ;Get char
C120:5A        27         phy
C121:BD B8 04  28         lda   pwidth,x   ;Formatting enabled?
C124:F0 42    C168  29         beq   prnow
C126:A5 24     30         lda   ch         ;Get current horiz position
C128:B0 1C    C146  31         bcs   servid  ;Branch if video echo
C12A:DD B8 04  32         cmp   pwidth,x   ;If CH >= PWIDTH, then CH = COL
C12D:90 03    C132  33         bcc   chok
C12F:BD 38 07  34         lda   col,x
C132:DD 38 07  35 chok    cmp   col,x      ;Must be > col for valid tab
C135:B0 0B    C142  36         bcs   fixch   ;Branch if ok
C137:C9 11     37         cmp   #$11      ;8 or 16?
C139:B0 11    C14C  38         bcs   prnt    ;If > forget it
C138:09 F0     39         ora   #$F0      ;Find next comma cheaply
C13D:3D 38 07  40         and   col,x     ;Don't blame me it's Dick's trick
C140:65 24     41         adc   ch
C142:85 24     42 fixch   sta   ch         ;Save the new position
C144:80 06    C14C  43         bra   prnt
C146:C5 21     44 servid  cmp   wndwidth  ;If ch>= wndwidth go back to start of line
C148:90 02    C14C  45         blt   prnt
C14A:64 24     46         stz   ch         ;Go back to left edge

C14C:          48 * We have a char to print
C14C:7A        49 prnt   ply
C14D:5A        50         phy
C14E:BD 38 07  51         lda   col,x      ;Have we exceeded width?
C151:DD B8 04  52         cmp   pwidth,x
C154:B0 08    C15E  53         bge   toofar
C156:C5 24     54         cmp   ch         ;Are we tabbing?
C158:B0 0E    C168  55         bge   prnow
C15A:A9 40     56         lda   #$40      ;Space * 2
C15C:80 02    C160  57         bra   tab
C15E:A9 1A     58 toofar  lda   #$1A      ;CR * 2

```

C160:C0	80	59	tab	cpy	#\$80	;C = High bit
C162:6A		60		ror	A	;Shift it into char
C163:2D	9D C1	61		jsr	serout3	;Out it goes
C166:80	E4 C14C	62		bra	prnt	
C168:98		63	prnow	tya		
C169:20	8C C1	64		jsr	serout	;Print the actual char
C16C:BD	B8 04	65		lda	pwdth,x	;Formatting enabled
C16F:F0	17 C188	66		beq	done	
C171:3C	B8 06	67		bit	flags,x	;In video echo?
C174:30	12 C188	68		bml	done	
C176:BD	38 07	69		lda	col,x	;Check if within 8 chars of right edge
C179:FD	B8 04	70		sbc	pwdth,x	;So BASIC can format output
C17C:C9	F8	71		cmp	#\$F8	
C17E:90	04 C184	72		bcc	setch	;If not within 8, we're done
C180:18		73		clc		
C181:65	21	74		adc	wndwdth	
C183:AC		75		dfb	\$AC	;Dummy LDY to skip next two bytes
C184:A9	00	76	setch	lda	#0	;Keep cursor at 0 if video off
C186:85	24	77		sta	ch	
C188:68		78	done	pla		;Restore regs
C189:7A		79		ply		
C18A:FA		80		plx		
C18B:60		81	soemd	rte		

C18C:	C18C	83	serout	equ	*	;Serial output
C18C:2D	EB C9	84		jsr	command	;Check if command
C18F:9D	FA C18B	85		bcc	soemd	;All done if it is
C191:	C191	86	serout2	equ	*	
C191:3C	B8 06	87		bit	flags,x	;N=1 iff video on
C194:10	07 C19D	88		bpl	serout3	
C196:C9	91	89		cmp	#xon	;Don't echo ^Q
C198:F0	03 C19D	90		beq	serout3	
C19A:20	F0 FD	91		jsr	coul	;Echo it
C19D:	C19D	92	serout3	equ	*	
C19D:BC	85 C8	93		ldy	devno,x	;Y points to ACIA
C1A0:48		94		pha		;Save the char
C1A1:2C	58 FF	95		bit	iorts	;Control char?
C1A4:F0	03 C1A9	96		beq	sordy	;Don't inc column if so
C1A6:FE	38 07	97		inc	col,x	
C1A9:08		98	sordy	php		;can't have real interrupts for a while
C1AA:78		99		sei		
C1AB:89	F9 BF	100		lda	sstat,y	;Check XMIT empty & DCD
C1AE:10	11 C1C1	101		bpl	sordy2	;branch if not clearing an interrupt
C1B0:48		102		pha		;save original status
C1B1:5A		103		phy		
C1B2:2C	14 C0	104		bit	rdramwrt	;Save state of aux ram
C1B5:08		105		php		
C1B6:20	1C C9	106		jsr	altst2	
C1B9:28		107		plp		
C1BA:10	03 C1BF	108		bpl	somain	;Branch if was main
C1BC:8D	05 C0	109		sta	wrcardram	;Was alt ram
C1BF:7A		110	somain	ply		
C1C0:68		111		pla		
C1C1:28		112	sordy2	plp		
C1C2:29	30	113		and	#\$30	
C1C4:C9	10	114		cmp	#\$10	

C1C6:D0 E1	C1A9	115	bne	sordy	
C1C8:68		116	pla		
C1C9:48		117	pha		;Get char to XMIT
C1CA:99 F8 BF		118	sta	sdata,y	;Out it goes
C1CD:3C B8 06		119	bit	Flags,x	;V=1 if LF after CR
C1D0:49 0D		120	eor	#\$0D	;check for CR.
C1D2:0A		121	asl	A	;preserve bit 7
C1D3:D0 0D	C1E2	122	bne	sodone	;branch if not CR.
C1D5:50 06	C1DD	123	bvc	clrcol	;branch if no LF after CR
C1D7:A9 14		124	lda	#\$14	;Get LF*2
C1D9:6A		125	ror	A	;no shift in high bit
C1DA:20 9D C1		126	jsr	serout3	;Output the LF but don't echo it
C1DD:64 24		127	stz	ch	;0 position & column
C1DF:9E 38 07		128	stz	col,x	
C1E2:68		129	pla		;Get the char back
C1E3:60		130	rts		

```

C1E4:          132 * Pascal support stuff

C1E4:48        134 plinit   pha
C1E5:20 C8 C2  135        jsr   default    ;set defaults, enable acia
C1E8:9E 88 06  136        stz   flags,x
C1EB:68        137        pla
C1EC:80 05    C1F3 138        bra   p1read2    ;all done...

C1EE:20 C5 C8  140 p1read   jsr   XRDSEr    ;read data from serial port (or buffer)
C1F1:90 FB    C1EE 141        bcc   p1read    ;Branch if data not ready
C1F3:A2 00    142 p1read2  ldx   #0
C1F5:60        143        rts

C1F6:20 8C C1  145 p1write  jsr   serout    ;Go output character
C1F9:80 F8    C1F3 146        bra   p1read2
C1FB:80 1A    C217 147 p1status bra  p2status

C1FD:          0003 149        ds     comslot-*, $00
C200:          19    INCLUDE COMM    ;Communications port @ $C200

```

C200:2C	58 FF	3	bit	iorts		;Set V to indicate initial entry
C203:70	28 C230	4	bvs	entr		
C205:38		5	sin	sec		;Input entry point
C206:90		6	dfb	\$90		;BCC opcode to skip next byte
C207:18		7	sout	clc		;Output entry point
C208:B8		8	clv			;Mark not initial entry
C209:50	25 C230	9	bvc	entr		;Branch around pascal entry stuff
C20B:01		11	dfb	\$01		;pascal signature byte
C20C:31		12	dfb	\$31		;device signature
C20D:11		13	dfb	>p2init		
C20E:13		14	dfb	>p2read		
C20F:15		15	dfb	>p2write		
C210:17		16	dfb	>p2status		
C211:		18	* Pascal support stuff			
C211:80	D1 C1E4	20	p2init	bra	plinit	
C213:80	D9 C1EE	21	p2read	bra	plread	
C215:80	DF C1F6	22	p2write	bra	plwrite	
C217:	C217	24	p2status	equ	*	
C217:A2	40	25	ldx	#\$40		;anticipate bad status request
C219:4A		26	lsr	a		;shift request to carry
C21A:D0	12 C22E	27	bne	notrdy		
C21C:AA		28	tax			;clear x for no error return code
C21D:A9	08	29	lda	#8		;anticipate input ready request
C21F:80	01 C222	30	bcs	pstat2		;branch if good guess.
C221:0A		31	asl	a		
C222:09	20	32	pstat2	ora	#\$20	;include DCD in test
C224:39	89 C0	33	and	sstat+\$90,y		
C227:F0	05 C22E	34	beq	notrdy		;branch if not ready for I/O
C229:49	20	35	eor	#\$20		
C22B:38		36	sec			;assume port is ready
C22C:D0	01 C22F	37	bne	lsrdy		;branch if good assumption
C22E:18		38	notrdy	clc		;indicate acia not ready for I/O
C22F:60		39	lsrdy	rts		
C230:DA		41	entr	phx		
C231:A2	C2	42	ldx	#\$comslot		;X = <CN00
C233:	C233	43	setup	equ	*	
C233:5A		44	phy			
C234:48		45	pha			
C235:8E	F8 07	46	stx	mslot		
C238:50	22 C25C	47	bvc	sudone		;First call?
C23A:A5	36	48	lda	cswl		;If both hooks CN00 setup defaults
C23C:45	38	49	eor	kswl		
C23E:F0	06 C246	50	beq	sudodef		
C240:A5	37	51	lda	cswh		;If both hooks CN then don't do def
C242:C5	39	52	cmp	kswh		;since it has already been done
C244:F0	03 C249	53	beq	sunodef		
C246:20	C8 C2	54	sudodef	jsr	default	;Set up defaults
C249:8A		55	sunodef	txa		
C24A:45	39	56	eor	kswh		;Input call?
C24C:05	38	57	ora	kswl		
C24E:D0	07 C257	58	bne	suout		;Must be Cn00
C250:A9	05	59	lda	#\$>sin		;Fix the input hook
C252:85	38	60	sta	kswl		

C254:38		61	sec		;C = 1 for input call
C255:80 05	C25C	62	bra	sudone	
C257:A9 07		63	suout	lda	#>out
C259:85 36		64	sta	caw1	;Fix output hook
C25B:18		65	clc		;Note C might not be 0
C25C:BD B8 06		66	sudone	lda	flags,x
C25F:89 01		67	bit	#1	;C=0 for output
C261:D0 03	C266	68	bne	comport	;Check if serial or comm port
C263:4C 17 C1		69	comout	jmp	serport
C266:90 FB	C263	70	comport	bcc	comout
C268:68		71	pla		;Output?
C269:80 28	C293	72	bra	term1	;Get the char
C26B:3C B8 03		73	noesc	bit	sermode,x
C26E:50 1C	C28C	74	bvc	exit1	;Input
C270:20 91 C1		75	jst	serout2	;In terminal mode?
C273:80 1E	C293	76	bra	term1	;If not, return key
C275:	C275	77	testkbd	equ	;Out it goes
C275:68		78	pla		
C276:20 70 CC		79	jsr	update	;Get current char
C279:10 1B	C296	80	bpl	serin	;Update cursor & check keyboard
C27B:20 EB C9		81	jsr	command	;N=0 if no new key
C27E:80 EB	C26B	82	bcs	noesc	;Test for command
C280:29 5F		83	and	#\$5F	;Branch if not
C282:C9 51		84	cmp	#'Q'	;upshift for following tests
C284:F0 04	C28A	85	beq	exitX	;Quit?
C286:C9 52		86	cmp	#'R'	
C288:D0 09	C293	87	bne	term1	;Reset?
C28A:A9 98		88	exitX	lda	#\$98
C28C:7A		89	exit1	ply	;Go check serial
C28D:FA		90	plx		;return a CTRL-X
C28E:60		91	rts		
C28F:18		92	goremode	clc	;Into remote mode
C290:20 CD CA		93	goterm	jsr	setterm
C293:	C293	94	term1	equ	;Into terminal mode
C293:20 4C CC		95	jsr	showcur	
C296:48		96	serin	pha	;Get current char on screen
C297:20 C5 C8		97	jsr	XRDSE	
C29A:90 D9	C275	98	bcc	testkbd	;Is it ready?
C29C:A8		99	tay		;If not, try the keyboard
C29D:68		100	pla		;Save new input in y for now
C29E:5A		101	phy		
C29F:20 B8 C3		102	jsr	storch	;Save new char on stack
C2A2:68		103	pla		;Fix the screen
C2A3:BC 38 06		104	ldy	eschar,x	;Get the new data
C2A6:F0 16	C2BE	105	beq	sinomod	;If 0, don't modify char
C2A8:09 80		106	ora	#\$80	
C2AA:C9 8A		107	cmp	#\$8A	;Apple loves the high bit
C2AC:F0 E5	C293	108	beq	term1	;Ignore line feed
C2AE:C9 91		109	cmp	#xon	
C2B0:F0 E1	C293	110	beq	term1	;Ignore ^Q
C2B2:C9 FF		111	cmp	#\$FF	;Ignore FFs
C2B4:F0 DD	C293	112	beq	term1	
C2B6:C9 92		113	cmp	#\$92	;^R for remote?
C2B8:F0 D5	C28F	114	beq	goremode	
C2BA:C9 94		115	cmp	#\$94	;^T for terminal mode?
C2BC:F0 D2	C290	116	beq	goterm	
C2BE:3C B8 03		117	sinomod	bit	sermode,x
C2C1:50 C9	C28C	118	bvc	exit1	;In terminal mode?
					;Return to user if not A = char

C2C3:20 ED FD	119	jsr	cout	;Onto the screen with it
C2C6:80 CB C293	120	bra	term1	
C2C8: C2C8	121 default	equ	*	;Set up the defaults
C2C8:20 A2 C8	122	jsr	moveirq	;make sure irq vectors ok
C2CB:BC 3B C2	123	ldy	defidx-\$C1,x	;Index into alt screen. Table in command
C2CE:20 7C C3	124 defloop	jsr	getalt	;Get default from alt screen
C2D1:48	125	pha		
C2D2:88	126	dey		
C2D3:30 04 C2D9	127	bmi	defff	;Done if minus
C2D5:C0 03	128	cpy	#3	
C2D7:D0 F5 C2CE	129	bne	defloop	;Or if 2
C2D9:20 A2 C8	130 defff	jsr	moveirq	;Jam irq vector into LC
C2DC:68	131	pla		;Command, control & flags on stack
C2DD:BC 85 C8	132	ldy	devno,x	
C2E0:99 FB BF	133	sta	sent1,y	;Set command reg
C2E3:68	134	pla		
C2E4:99 FA BF	135	sta	scomd,y	
C2E7:68	136	pla		
C2E8:9D B8 06	137	sta	flags,x	;And the flags
C2EB:29 01	138	and	#1	;A = \$01 (^A) if comm mode
C2ED:D0 02 C2F1	139	bne	defcom	
C2EF:A9 09	140	lda	#9	;^I for serial port
C2F1:9D 38 06	141 defcom	sta	eschar,x	
C2F4:68	142	pla		;Get printer width
C2F5:9D B8 04	143	sta	pwdth,x	
C2F8:9E B8 03	144	stz	sermode,x	
C2FB:60	145	rts		
C2FC:03 07	146 defidx	dfb	3,7	
C2FE: 0002	147	ds	\$C300-*, \$00	
C300:	20	INCLUDE	C3SPACE	;80 column card @ \$C300



```

C300:      2 *****
C300:      3 *
C300:      4 * THIS IS THE $C3XX ROM SPACE:
C300:      5 *
C300:      6 *****
C300:48     7 C3ENTRY   PHA           ;save regs
C301:DA     8         PHX
C302:5A     9         PHY
C303:80 12  C317   10        BRA    BASICINIT    ;and init video firmware
C305:38     11 C3KEYIN SEC           ;Pascal 1.1 ID byte
C306:90     12        DFB    $90           ;BCC OPCODE (NEVER TAKEN)
C307:18     13 C3COUTI CLC           ;Pascal 1.1 ID byte
C308:80 1A  C324   14        BRA    BASICENT    ;=>go print/read char
C30A:EA     15        NOP
C30B:      16 *
C30B:      17 * PASCAL 1.1 FIRMWARE PROTOCOL TABLE:
C30B:      18 *
C30B:01     19        DFB    $01           ;GENERIC SIGNATURE BYTE
C30C:88     20        DFB    $88           ;DEVICE SIGNATURE BYTE
C30D:      21 *
C30D:2C     22        DFB    >JPINIT    ;PASCAL INIT
C30E:2F     23        DFB    >JPREAD    ;PASCAL READ
C30F:32     24        DFB    >JPWRITE   ;PASCAL WRITE
C310:35     25        DFB    >JPSTAT    ;PASCAL STATUS
C311:      26 *****
C311:      27 *
C311:      28 * 128K SUPPORT ROUTINE ENTRIES:
C311:      29 *
C311:4C 86 CF   30        JMP    MOVEAUX    ;MEMORY MOVE ACROSS BANKS
C314:4C CD CF   31        JMP    XFER      ;TRANSFER ACROSS BANKS
C317:      32 *****
C317:      33 *
C317:      34 *****
C317:      35 * BASIC I/O ENTRY POINT:
C317:      36 *****
C317:      37 *
C317:20 20 CE   38 BASICINIT JSR    HOOKUP    ;COPYROM if needed, sethooks
C31A:20 BE CD   39        JSR    SET80     ;setup 80 columns
C31D:20 58 FC   40        JSR    HOME     ;clear screen
C320:7A     41        PLY
C321:FA     42        PLX           ;restore X
C322:68     43        PLA           ;restore char
C323:18     44        CLC           ;output a character
C324:      45 *
C324:80 03  C329   46 BASICENT BCS    BINPUT    ;=>carry me to input
C326:4C F6 FD   47 BPRINT  JMP    COUTZ     ;print a character
C329:4C 1B FD   48 BINPUT  JMP    KEYIN     ;get a keystroke
C32C:      49 *
C32C:4C 41 CF   50 JPINIT  JMP    PINIT     ;pascal init
C32F:4C 35 CF   51 JPREAD  JMP    PASREAD    ;pascal read
C332:4C 02 CE   52 JPWRITE JMP    PWRITE     ;pascal write
C335:4C B1 CE   53 JPSTAT  JMP    PSTATUS    ;pascal status call
C338:      54 *
C338:      55 * COPYROM is called when the video firmware is
C338:      56 * initialized. If the language card is switched
C338:      57 * In for reading, it copies the FB ROM to the
C338:      58 * language card and restores the state of the
C338:      59 * language card.

```

```

C338:          60 *
C338:A9 D6     61 COPYROM LDA #GOODF8 ;get the ID byte
C33A:          62 *
C33A:          63 * Compare ID bytes to whatever is readable. If it
C33A:          64 * matches, all is ok. If not, need to copy.
C33A:          65 *
C33A:CD 83 FB   66 CMP F8VERSION ;does it match?
C33D:FD 3C C37B 67 BEQ ROMOK
C33F:20 60 C3   68 JSR SETROM ;read ROM, write RAM, save state
C342:A9 F8     69 LDA #$F8 ;from F800-FFFF
C344:85 37     70 STA CSWH
C346:64 36     71 STZ CSWL
C348:B2 36     72 COPYROM2 LDA (CSWL) ;get a byte
C34A:92 36     73 STA (CSWL) ;and save a byte
C34C:E6 36     74 INC CSWL
C34E:D0 F8 C348 75 BNE COPYROM2
C350:E6 37     76 INC CSWH
C352:D0 F4 C348 77 BNE COPYROM2 ;fall into RESETLC
C354:          78 *
C354:          79 * RESETLC resets the language card to the state
C354:          80 * determined by SETROM. It always leaves the card
C354:          81 * write enabled.
C354:          82 *
C354:DA        83 RESETLC PHX ;save X
C355:AE 78 04   84 LDX ROMSTATE ;get the state
C358:3C 81 C0   85 BIT ROMIN,X ;set bank & ROM/RAM read
C35B:3C 81 C0   86 BIT ROMIN,X ;set write enable
C35E:FA        87 PLX ;restore X
C35F:60        88 RTS
C360:          89 *
C360:          90 * SETROM switches in the ROM for reading, the RAM
C360:          91 * for writing, and it saves the state of the
C360:          92 * language card. It does not save the write
C360:          93 * protect status of the card.
C360:          94 *
C360:DA        95 SETROM PHX ;save X
C361:A2 00     96 LDX #0 ;assume write enable,bank2,ROMRD
C363:2C 11 C0   97 BIT RDLCSBK2 ;is bank 2 switched in?
C366:30 02 C36A 98 BMI NOT1 ;=>yes
C368:A2 08     99 LDX #$8 ;indicate bank 1
C36A:2C 12 C0 100 NOT1 BIT RDLGRAM ;is LC RAM readable?
C36D:10 02 C371 101 BPL NOREAD ;=>no
C36F:E8       102 INX ;indicate RAM read
C370:E8       103 INX
C371:2C 81 C0 104 NOREAD BIT $C081 ;ROM read
C374:2C 81 C0 105 BIT $C081 ;RAM write
C377:8E 78 04 106 STX ROMSTATE ;save state
C37A:FA       107 PLX ;restore X
C37B:60       108 ROMOK RTS
C37C:          109 *
C37C:          110 * GETALT reads a byte from aux memory screenholes.
C37C:          111 * Y is the index to the byte (0-7) indexed off of
C37C:          112 * address $478.
C37C:          113 *
C37C:AD 13 C0 114 GETALT LDA RDRAMRD ;save state of aux memory
C37F:0A       115 ASL A
C380:AD 18 C0 116 LDA RD80COL ;and of the 80STORE switch
C383:08       117 PHP

```

C384:8D 00 C0	118	STA	CLR80COL	;no 80STORE to get page 1
C387:8D 03 C0	119	STA	RDCARDRAM	;pop in the other half of RAM
C38A:B9 78 04	120	LDA	\$478,Y	;read the desired byte
C38D:28	121	PLP		;and restore memory
C38E:B0 03 C393	122	BCS	GETALT1	
C390:8D 02 C0	123	STA	RDMAINRAM	
C393:10 03 C398	124	BPL	GETALT2	
C395:8D 01 C0	125	STA	SET80COL	
C398:60	126	GETALT2	RTS	
C399:	127	*		
C399:09 B0	128	UPSHIFT0	ORA	#\$80 ;set high bit for execs
C39B:C9 FB	129	UPSHIFT	CMP	#\$FB
C39D:B0 06 C3A5	130	BCS	X.UPSHIFT	
C39F:C9 E1	131	CMP	#\$E1	
C3A1:90 02 C3A5	132	BCC	X.UPSHIFT	
C3A3:29 DF	133	AND	#\$DF	
C3A5:60	134	X.UPSHIFT	RTS	
C3A6:	135	*		
C3A6:	136	*	GETCOUT	performs COUT for GETLN. It disables the
C3A6:	137	*		echoing of control characters by clearing the
C3A6:	138	*	M.CTL	mode bit, prints the char, then restores
C3A6:	139	*	M.CTL.	NOESC is used by the RDKEY routine to
C3A6:	140	*		disable escape sequences.
C3A6:	141	*		
C3A6:48	142	GETCOUT	PHA	;save char to print
C3A7:A9 08	143	LDA	%M.CTL	;disable control chars
C3A9:1C FB 04	144	TRB	VMODE	;by clearing M.CTL
C3AC:68	145	PLA		;restore character
C3AD:20 ED FB	146	JSR	COUT	;and print it
C3B0:4C 44 FD	147	JMP	NOESCAPE	;enable control chars
C3B3:	148	*		
C3B3:	149	*	STORCH	determines loads the current cursor position,
C3B3:	150	*		inverts the character, and displays it
C3B3:	151	*	STORCHAR	inverts the character and displays it at the
C3B3:	152	*		position stored in Y
C3B3:	153	*	STORY	determines the current cursor position, and
C3B3:	154	*		displays the character without inverting it
C3B3:	155	*	STORE	displays the char at the position in Y
C3B3:	156	*		
C3B3:	157	*		If mouse characters are enabled (VMODE bit 0 = 0)
C3B3:	158	*		then mouse characters (\$40-\$5F) are displayed when
C3B3:	159	*		the alternate character set is switched in. Normally
C3B3:	160	*		values \$40-\$5F are shifted to \$0-\$1F before display.
C3B3:	161	*		
C3B3:	162	*		Calls to GETCUR trash Y
C3B3:	163	*		
C3B3:20 9D CC	164	STORY	JSR	GETCUR ;get newest cursor into Y
C3B6:80 09 C3C1	165	BRA	STORE	
C3B8:	166	*		
C3B8:20 9D CC	167	STORCH	JSR	GETCUR ;first, get cursor position
C3BB:24 32	168	BIT	INVFLG	;normal or inverse?
C3BD:30 02 C3C1	169	BMI	STORE	;=>normal, store it
C3BF:29 7F	170	AND	#\$7F	;inverse it
C3C1:5A	171	STORE	PHY	;save real Y
C3C2:09 00	172	ORA	#0	;does char have high bit set?
C3C4:30 15 C3DB	173	BMI	STORE1	;=>yes, don't do mouse check
C3C6:48	174	PHA		;save char
C3C7:AD FB 04	175	LDA	VMODE	;is mouse bit set?

C3CA:6A		176	ROR	A	
C3CB:68		177	PLA		;restore char
C3CC:90 0D C3DB		178	BCC	STORE1	;=>no, don't do mouse shift
C3CE:2C 1E C0		179	BIT	ALTCHARSET	;no shift if    char set
C3D1:10 08 C3DB		180	BPL	STORE1	;=> it is!
C3D3:49 40		181	EOR	#\$40	;\$40-\$5F=>0-\$1f
C3D5:89 60		182	BIT	#\$60	
C3D7:F0 02 C3DB		183	BEQ	STORE1	
C3D9:49 40		184	EOR	#\$40	
C3DB:2C 1F C0		185	BIT	RD80VID	;80 columns?
C3DE:10 19 C3F9		186	BPL	STORE5	;=>no, store char
C3E0:48		187	PHA		;save (shifted) char
C3E1:8D 01 C0		188	STA	SET80COL	;hit 80 store
C3E4:98		189	TYA		;get proper Y
C3E5:45 20		190	EOR	WNDLFT	C=1 if char in main ram
C3E7:4A		191	LSR	A	
C3E8:B0 04 C3EE		192	BCC	STORE2	;=>yes, main RAM
C3EA:AD 55 C0		193	LDA	TXTPAGE2	;else flip in aux RAM
C3ED:C8		194	INY		;do this for odd left, aux bytes
C3EE:98		195	TYA		;divide pos'n by 2
C3EF:4A		196	LSR	A	
C3F0:A8		197	TAY		
C3F1:68		198	PLA		;get (shifted) char
C3F2:91 2B		199	STA	(BASL),Y	;stuff it
C3F4:2C 54 C0		200	BIT	TXTPAGE1	;else restore page1
C3F7:7A		201	PLY		;restore real Y
C3F8:60		202	RTS		;und exit
C3F9:		203	*		
C3F9:91 2B		204	STA	(BASL),Y	;do 40 column store
C3FB:7A		205	PLY		;restore Y
C3FC:60		206	RTS		;and exit
C3FD:	0003	207	DS	SC400-*,500	
C400:		21	INCLUDE	MOUSE	;Equates for the mouse

```

C400:          2          MSB   ON
C400:          3 *****
C400:          4 *
C400:          5 * Mouse firmware for the Chels
C400:          6 *
C400:          7 *   by Rich Williams
C400:          8 *   July, 1983
C400:          9 *
C400:         10 *****

C400:         12 *****
C400:         13 *
C400:         14 * Equates
C400:         15 *
C400:         16 *****

C400:          18 * Input bounds are in scratch area
C400:    0478    19 moutemp equ $478          ;Temporary storage
C400:    0478    20 minl   equ $478
C400:    04F8    21 maxl   equ $4F8
C400:    0578    22 minh   equ $578
C400:    05F8    23 maxh   equ $5F8
C400:          24 * Mouse bounds in slot 5 screen area
C400:    047D    25 minxl   equ $47D
C400:    04FD    26 minyl   equ $4FD
C400:    057D    27 minxh   equ $57D
C400:    05FD    28 minyh   equ $5FD
C400:    067D    29 maxx1   equ $67D
C400:    06FD    30 maxyl   equ $6FD
C400:    077D    31 maxxh   equ $77D
C400:    07FD    32 maxyh   equ $7FD
C400:          33 * Mouse holes in slot 4 screen area
C400:    047C    34 mouxl   equ $47C          ;X position low byte
C400:    04FC    35 mouyl   equ $4FC          ;Y position low byte
C400:    057C    36 mouxh   equ $57C          ;X position high byte
C400:    05FC    37 mouyh   equ $5FC          ;Y position high byte
C400:    067C    38 mouarm   equ $67C          ;Arm interrupts from movement or button
C400:    077C    39 moustat  equ $77C          ;Mouse status
C400:          40 * Moustat provides the following
C400:          41 * D7= Button pressed
C400:          42 * D6= Status of button on last read
C400:          43 * D5= Moved since last read
C400:          44 * D4= Reserved
C400:          45 * D3= Interrupt from VBL
C400:          46 * D2= Interrupt from button
C400:          47 * D1= Interrupt from movement
C400:          48 * D0= Reserved
C400:    07FC    49 mousmode  equ $7FC          ;Mouse mode
C400:          50 * D7-D4= Unused
C400:          51 * D3= VBL active
C400:          52 * D2= VBL interrupt on button
C400:          53 * D1= VBL interrupt on movement
C400:          54 * D0= Mouse active
C400:    0020    55 movarm   equ $20

```

```

C400:      000C  56 vblmode equ  $0C
C400:      0004  57 butmode equ  $04      ;D2 mask
C400:      0002  58 movmode equ  $02      ;D1 mask


C400:      60 * Hardware addresses
C400:      C015  61 mouxint equ  $C015      ;D7 = x interrupt
C400:      C017  62 mouyint equ  $C017      ;D7 = y interrupt
C400:      C019  63 vblint equ  $C019      ;D7 = vbl interrupt
C400:      C078  64 ioudsbl equ  $C078      ;Disable iou access
C400:      C079  65 iouenbl equ  $C079      ;Enable iou access
C400:      C048  66 mouclr equ  $C048      ;Clear mouse interrupt
C400:      C058  67 iou equ  $C058      ;IOU interrupt switches
C400:      C058  68 moudsbl equ  $C058      ;Disable mouse interrupts
C400:      C059  69 mouenbl equ  $C059      ;Enable mouse interrupts
C400:      C063  70 moubut equ  $C063      ;D7 = Mouse button
C400:      C066  71 mouxl equ  $C066      ;D7 = X1
C400:      C067  72 mouyl equ  $C067      ;D7 = Y1
C400:      C070  73 vblclr equ  $C070      ;Clear VBL interrupt
C400:      74 *
C400:      75 * Other addresses
C400:      76 *
C400:      0200  77 inbuf equ  $200      ;Input buffer
C400:      0214  78 binl equ  inbuf+20      ;Temp for binary conversion
C400:      0215  79 binh equ  inbuf+21
C400:      22      INCLUDE MCODE      ;Mouse @ $C400

```



```

C400:      2 *****
C400:      3 *
C400:      4 * Entry points for mouse firmware
C400:      5 *
C400:      6 *****
C400:80 05   C407 7 mbasic   bra   outent
C402:A2 03   8 pnull    ldx   #3
C404:60      9          rts           ;Null for pascal entry
C405:38     10 inent    sec           ;Signature bytes
C406:90     11          dfb   $90
C407:18     12 outent   cle
C408:4C 80 C7 13          jmp   xmbasic   ;Go do basic entry
C40B:01     14          dfb   $01         ;More signature stuff
C40C:20     15          dfb   $20
C40D:02     16          dfb   >pnull
C40E:02     17          dfb   >pnull
C40F:02     18          dfb   >pnull
C410:02     19          dfb   >pnull
C411:00     20          dfb   $0
C412:3D     21          dfb   >xsetmou   ;SETMOUSE
C413:FC     22          dfb   >xmtstint  ;SERVENOUSE
C414:95     23          dfb   >xmread   ;READMOUSE
C415:84     24          dfb   >xmclear  ;CLEARMOUSE
C416:6B     25          dfb   >noerror  ;POSMOUSE
C417:BD     26          dfb   >xmclamp  ;CLAMPMOUSE
C418:6D     27          dfb   >xmhome   ;HOMEMOUSE
C419:1C     28          dfb   >initmouse ;INITMOUSE
C41A:02     29          dfb   >pnull
C41B:CF     30          dfb   >xmint

```



```

C41C:          32 *****
C41C:          33 *
C41C:          34 * Initmouse - resets the mouse
C41C:          35 * Also clears all of the mouse holes
C41C:          36 * note that iou access fires pdlstrb & makes mouse happy
C41C:          37 *
C41C:          38 *****
C41C:          39 initmouse equ *
C41C:9C 7C 07    40          stz     moustat      ;Clear status
C41F:A2 80      41          ldx     #$80
C421:A0 01      42          ldy     #1
C423:9E 7D 04   43 xrlong    stz     minxl,x      ;Minimum = $0000
C426:9E 7D 05   44          stz     minxh,x
C429:A9 FF      45          lda     #$FF          ;Maximum = $03FF
C42B:9D 7D 06   46          sta     maxxl,x
C42E:A9 03      47          lda     #03
C430:9D 7D 07   48          sta     maxxh,x
C433:A2 00      49          ldx     #0
C435:88         50          dey
C436:10 EB C423 51          bpl     xrlong
C438:2D 6D C4    52          jsr     xhome      ;Clear the mouse holes
C43B:A9 00      53          lda     #0          ;Fall into SETMOU

C43D:          55 *****
C43D:          56 *
C43D:          57 * XSETMOU - Sets the mouse mode to A
C43D:          58 *
C43D:          59 *****
C43D:          60 xsetmou equ *
C43D:AA         61          tax
C43E:2D A2 C8    62          jsr     moveirq      ;Make sure interrupt vector is right
C441:8A         63          txa          ;Only x preserved by moveirq
C442:8D 78 04    64          sta     moutemp
C445:4A         65          lsr     A          ;D0 = 1 if mouse active
C446:0D 78 04    66          ora     moutemp      ;D2 = 1 if vbl active
C449:C9 10      67          cmp     #$10        ;If >=$10 then invalid mode
C44B:B0 1F C46C 68          bcs     sminvalid
C44D:29 05      69          and     #5          ;Extract VBL & Mouse
C44F:F0 01 C452 70          beq     xsoff      ;Turning it off?
C451:58         71          cli          ;If not, ints active
C452:69 55      72 xsoff     adc     #$55      ;Make iou byte C=0

C454:          74 *****
C454:          75 *
C454:          76 * SETIOU - Sets the IOU interrupt modes to A
C454:          77 * Inputs: A = Bits to change
C454:          78 * D7 = Y int on falling edge
C454:          79 * D6 = Y int on rising edge
C454:          80 * D5 = X int on falling edge
C454:          81 * D4 = X int on rising edge
C454:          82 * D3 = Enable VBL int
C454:          83 * D2 = Disable VBL int
C454:          84 * D1 = Enable mouse int
C454:          85 * D0 = Disable mouse int

```

```

C454:      86 *
C454:      87 *
C454:      88 *****
C454:      C454 89 setiou equ *
C454:08      90 php
C455:78      91 sei ;Don't allow ints while iou enabled
C456:8E FC 07 92 stx moumode
C459:8D 79 C0 93 sta iouenbl ;Enable iou access
C45C:A2 08    94 ldx #8
C45E:CA      95 siloop dex
C45F:0A      96 asl A ;Get a bit to check
C460:90 03 C465 97 bcc sinoch ;No change if C=0
C462:9D 58 C0    98 sta iou,x ;Set it
C465:D0 F7 C45E 99 sinoch bne siloop ;Any bits left in A?
C467:8D 78 C0   100 sta ioudsbl ;Turn off iou access
C46A:28        101 plp
C46B:18        102 noerror clc
C46C:60        103 sminvalid rts

```

```

C46D:      105 *****
C46D:      106 *
C46D:      107 * XMHOME- Clears mouse position & status
C46D:      108 *
C46D:      109 *****
C46D:      C46D 110 xmhome equ *
C46D:A2 80      111 ldx #$80 ;Point mouse to upper left
C46F:80 02 C473 112 bra xmh2
C471:A2 00      113 xmhloop ldx #0
C473:BD 7D 04   114 xmh2 lda minxl,x
C476:9D 7C 04   115 sta mouxl,x
C479:BD 7D 05   116 lda minxh,x
C47C:9D 7C 05   117 sta mouxh,x
C47F:CA        118 dex
C480:10 EF C471 119 bpl xmhloop
C482:80 0C C490 120 bra xmdone

```

```

C484:      122 *****
C484:      123 *
C484:      124 * XMCLEAR - Sets the mouse to 0,0
C484:      125 *
C484:      126 *****
C484:      C484 127 xmclear equ *
C484:9C 7C 04   128 stz mouxl
C487:9C 7C 05   129 stz mouxh
C48A:9C FC 04   130 stz mouyl
C48D:9C FC 05   131 stz mouyh
C490:9C 7C 06   132 xmdone stz mouarm
C493:18        133 clc
C494:60        134 rts

```

```

C495:          136 *****
C495:          137 *
C495:          138 * XMREAD - Updates the screen holes
C495:          139 *
C495:          140 *****
C495:          C495 141 xmread    equ    *
C495:A9 20      142          lda    #movarm    ;Has mouse moved?
C497:2D 7C 06    143          and    mouarm
C49A:1C 7C 06    144          trb    mouarm    ;Clear arm bit
C49D:2C 63 C0    145          bit    moubut    ;Button pressed?
C4A0:30 02      C4A4 146          bml    xrbut
C4A2:09 80      147          ora    #$80
C4A4:2C 7C 07    148 xrbut     bit    moustat    ;Pressed last time?
C4A7:10 02      C4AB 149          bpl    xrbut2
C4A9:09 40      150          ora    #$40
C4AB:8D 7C 07    151 xrbut2    sta    moustat
C4AE:18          152          clc
C4AF:60          153          rts

```

```

C4B0:          155 *****
C4B0:          156 *
C4B0:          157 * XMCLAMP - Store new bounds
C4B0:          158 * Inputs A = 1 for Y, 0 for X axis
C4B0:          159 * minl, minh, maxl, maxh = new bounds
C4B0:          160 *
C4B0:          161 *****
C4B0:          C4B0 162 xmclamp   equ    *
C4B0:6A          163          ror    A            ;1 -> 80
C4B1:6A          164          ror    A
C4B2:29 80      165          and    #$80
C4B4:AA          166          tax
C4B5:AD 78 04    167          lda    minl
C4B8:9D 7D 04    168          sta    minxl,x
C4BB:AD 78 05    169          lda    minh
C4BE:9D 7D 05    170          sta    minxh,x
C4C1:AD F8 04    171          lda    maxl
C4C4:9D 7D 06    172          sta    maxxh,x
C4C7:AD F8 05    173          lda    maxh
C4CA:9D 7D 07    174          sta    maxxh,x
C4CD:18          175          clc            ;No error
C4CE:60          176          rts

```

```

C4CF:          178 *****
C4CF:          179 *
C4CF:          180 * Mouse interrupt handler
C4CF:          181 *
C4CF:          182 * MOUSEINT - Monitor's interrupt handler
C4CF:          183 * XMINT - Interrupt handler the user can use
C4CF:          184 * XMTSTINT - Checks mouse status bits
C4CF:          185 *****
C4CF:          C4CF 186 xmint      equ      *
C4CF:AE 66 C0    187      ldx      mouxl      ;Get X1 & Y1 asap
C4D2:AC 67 C0    188      ldy      mouyl
C4D5:          C4D5 189 mouseint   equ      *      ;Entry point if X & Y set up
C4D5:A9 0E       190      lda      #$0E      ;Clear status bits
C4D7:1C 7C 07    191      trb      moustat

C4DA:38          193      sec                      ;Assume interrupt not handled
C4DB:          194 * Check for vertical blanking interrupt
C4DB:AD 19 C0    195      lda      vblint      ;VBL interrupt?
C4DE:10 48      C528 196      bpl      chk mou
C4E0:8D 79 C0    197      sta      iouenbl      ;Enable iou access & clear VBL interrupt
C4E3:A9 0C       198      lda      #vblmode      ;Should we leave vbl active?
C4E5:2C FC 07    199      bit      moumode
C4E8:D0 03      C4ED 200      bne      cvnovbl
C4EA:8D 5A C0    201      sta      iou+2      ;Disable VBL
C4ED:09 02       202      cvnovbl   ora      #movmode
C4EF:80 1B      C50C 203      bra      xmskip

C4F1:A9 0E       205      mistat   lda      #$0E
C4F3:2D 7C 07    206      and      moustat
C4F6:D0 01      C4F9 207      bne      nostat2
C4F8:38          208      sec
C4F9:68          209      nostat2   pla
C4FA:60          210      rts
C4FB:          0000 211      ds      $C4FB-*
C4FB:D6          212      dfb      $D6      ;Signature byte
C4FC:48          213      xmtstint  pha
C4FD:18          214      clc
C4FE:80 F1      C4F1 215      bra      mistat      ;Go check status
C500:FF          216      dfb      $FF
C501:20 4D CE    217      jsr      zzquit      ;Get out of the hooks
C504:A2 FF       218      ldx      #$FF
C506:20 24 CB    219      qloop     jsr      zznrm2
C509:10 FB      C506 220      bpl      qloop
C50B:60          221      rts
C50C:          C50C 222      xmskip   equ      *
C50C:8D 78 C0    223      sta      ioudbl
C50F:2C 7C 06    224      bit      mouarm      ;VBL bit in arm isn't used
C512:D0 02      C516 225      bne      cvmoved
C514:A9 0C       226      lda      #vblmode      ;Didn't move
C516:2C 63 C0    227      cvmoved   bit      moubut      ;Button pressed?
C519:10 02      C51D 228      bpl      cvbut
C51B:49 04       229      eor      #butmode      ;Clear the button bit
C51D:2D FC 07    230      cvbut     and      moumode      ;Which bits were set in the mode
C520:0C 7C 07    231      tsb      moustat

```

C523:1C 7C 06	232	trb	mouarm	
C526:69 FE	233	adc	#\$FE	;C=1 if int passes to user
C528:	234	* Check & update mouse movement		
C528:	235	chk mou	equ	*
C528:AD 15 C0	236	lda	mouxint	;Mouse interrupt?
C52B:0D 17 C0	237	ora	mouylnt	
C52E:1C 6A C59A	238	bpl	xmdone	;If not return with C from vbl
C530:8A	239	txa		;Get X1 in A
C531:A2 00	240	ldx	#0	
C533:2C 15 C0	241	bit	mouxint	;X movement?
C536:30 0A C542	242	bmi	cmxmov	
C538:98	243	cmloop	tya	;Get Y1 into A
C539:49 80	244	eor	#\$80	;Complement direction
C53B:A2 80	245	ldx	#\$80	
C53D:2C 17 C0	246	bit	mouyint	
C540:1C 39 C57B	247	bpl	cmnoy	
C542:0A	248	cmxmov	asl	A
C543:BD 7C 04	249	lda	mouxl,x	;A = current low byte
C546:B0 1A C562	250	bcs	cmrght	;Which way?
C548:DD 7D 04	251	cmp	minxl,x	;Move left
C54B:DD 08 C555	252	bne	cmlok	
C54D:BD 7C 05	253	lda	mouxh,x	
C550:DD 7D 05	254	cmp	minxh,x	
C553:F0 22 C577	255	beq	cmnoint	
C555:BD 7C 04	256	cmlok	lda	mouxl,x
C558:DD 03 C55D	257	bne	cmnt0	;Borrow from high byte?
C55A:DE 7C 05	258	dec	mouxh,x	
C55D:DE 7C 04	259	cmnt0	dec	mouxl,x
C560:8C 15 C577	260	bra	cmnoint	
C562:DD 7D 06	261	cmrght	cmp	maxxl,x
C565:DD 08 C56F	262	bne	cmrok	;At high bound?
C567:BD 7C 05	263	lda	mouxh,x	
C56A:DD 7D 07	264	cmp	maxxh,x	
C56D:F0 08 C577	265	beq	cmnoint	
C56F:FE 7C 04	266	cmrok	inc	mouxl,x
C572:DD 03 C577	267	bne	cmnoint	;Move right
C574:FE 7C 05	268	inc	mouxh,x	
C577:E0 00	269	cmnoint	cpx	#0
C579:F0 8D C538	270	beq	cmloop	
C57B:8D 48 C0	271	cmnoy	sta	mouclr
C57E:A9 02	272	lda	#movmode	;Should we enable VBL?
C580:2D FC 07	273	and	moumode	
C583:F0 09 C58E	274	beq	cmnovbl	;Branch if not
C585:8D 79 C0	275	sta	iouenbl	
C588:8D 5B C0	276	sta	iou+3	;Enable VBL int
C58B:8D 78 C0	277	sta	ioudsbl	
C58E:09 20	278	cmnovbl	ora	#movarm
C590:0C 7C 06	279	tsb	mouarm	;Mark that we moved
C593:A9 0E	280	lda	#\$0E	
C595:2D 7C 07	281	and	moustat	
C598:69 FE	282	adc	#\$FE	;C=1 iff any bits were 1
C59A:60	283	xmdone	rts	

```

C59B:      285 *****
C59B:      286 *
C59B:      287 * HEXTODEC - Puts +0000, into the input buffer
C59B:      288 * inputs: A = Low byte of number
C59B:      289 *          X = High byte of number
C59B:      290 *          Y = Position of ones digit
C59B:      291 *
C59B:      292 *****
C59B:      293 hextoDEC equ *
C59B:EO 80 C59B 294 cpx #80 ;Is it a negative number?
C59D:90 0D C5AC 295 bcc hexdec2
C59F:49 FF 296 eor #FFF ;Form two's complement
C5A1:69 0D 297 adc #0 ;C = 1 from compare
C5A3:48 298 pha ;Save it
C5A4:8A 299 txa
C5A5:49 FF 300 eor #FFF
C5A7:69 0D 301 adc #0
C5A9:AA 302 tax
C5AA:68 303 pla
C5AB:38 304 sec
C5AC:8D 14 02 305 hexdec2 sta binl ;Store the number to convert
C5AF:8E 15 02 306 stx binh
C5B2:A9 AB 307 lda #'+' ;Store the sign in the buffer
C5B4:90 02 C588 308 bcc hdp0s2
C5B6:A9 AD 309 lda #'-'
C5B8:48 310 hdp0s2 pha ;Save the sign
C5B9:A9 AC 311 lda #',' ;Store a comma after the number
C5BB:99 01 02 312 sta inbuf+1,y
C5BE: C5BE 313 hdl00p equ * ;Divide by 10
C5BE: 314 *
C5BE: 315 * Divide BINH,L by 10 and leave remainder in A
C5BE: 316 *
C5BE:A2 11 317 ldx #16+1 ;16 bits and first time do nothing
C5C0:A9 0D 318 lda #0
C5C2:18 319 clc ;C=0 so first ROL leaves A=0
C5C3:2A 320 dvl0loop rol A
C5C4:C9 0A 321 cmp #10 ;A >= 10?
C5C6:90 02 C5CA 322 bcc dvl0lt ;Branch if <
C5C8:E9 0A 323 sbc #10 ;C = 1 from compare and is left set
C5CA:2E 14 02 324 dvl0lt rol binl
C5CD:2E 13 02 325 rol binh
C5D0:CA 326 dex
C5D1:D0 FD C5C3 327 bne dvl0loop
C5D3:09 B0 328 ora #'0' ;Make a ascii char
C5D5:99 0D 02 329 sta inbuf,y
C5D8:88 330 dey
C5D9:F0 D8 C5E3 331 beq hddone ;Stop on 0,6,12
C5DB:C0 D7 332 cpy #7
C5DD:F0 04 C5E3 333 beq hddone
C5DF:C0 0E 334 cpy #14
C5E1:D0 D8 C5BE 335 bne hdl00p
C5E3:68 336 hddone pla ;Get the sign
C5E4:99 0D 02 337 sta inbuf,y
C5E7:60 338 rts
C5E8:DF 67 37 1C 339 qtbl dfb $DF,$67,$37,$1C,$07,$0C,$45,$62
C5F0:6E 7E 38 0A 340 dfb $6E,$7E,$3B,$0A,$0B,$48,$77,$7B
C5F8:66 2B 0C 08 341 dfb $66,$2B,$0C,$0B,$16,$53,$68,$C5
C600: 0000 342 ds $C600-*

```



```

C600:      0356   3 DNIBL    EQU    $356
C600:      0300   4 NBUF1    EQU    $300
C600:      07DB   5 BOOTSCRN EQU    $7DB
C600:      002B   6 SLOTZ    EQU    $2B
C600:      003C   7 BOOTTMP   EQU    $3C
C600:      004F   8 BOOTDEV   EQU    $4F
C600:A2 20      9          LDX    #$20
C602:A0 00     10          LDY    #$00
C604:64 03     11          STZ    $03
C606:64 3C     12          STZ    $3C
C608:A9 60     13          LDA    #$60
C60A:AA        14          TAX
C60B:86 2B     15 DRV2ENT   STX    SLOTZ
C60D:85 4F     16          STA    BOOTDEV
C60F:5A        17          PHY
C610:BD 8E C0   18          LDA    $C08E,X
C613:BD 8C C0   19          LDA    $C08C,X
C616:7A        20          PLY
C617:B9 EA C0   21          LDA    $C0EA,Y
C61A:BD 89 C0   22          LDA    $C089,X
C61B:A0 50     23          LDY    #$50
C61F:BD 80 C0   24 SEEKZERO  LDA    $C080,X
C622:98        25          TYA
C623:29 03     26          AND    #$03
C625:0A        27          ASL    A
C626:05 2B     28          ORA    SLOTZ
C628:AA        29          TAX
C629:BD 81 C0   30          LDA    $C081,X
C62C:A9 56     31          LDA    #$56
C62E:20 AB FC   32          JSR    WAIT
C631:88        33          DEY
C632:10 EB C61F 34          BPL    SEEKZERO
C634:85 26     35          STA    $26
C636:85 3D     36          STA    $3D
C638:85 41     37          STA    $41
C63A:20 09 C7   38          JSR    MAKIBL
C63D:64 03     39 EXTENT1   STZ    $03
C63F:18        40 RDADR    CLC
C640:08        41          PHP
C641:28        42 RETRY1    PLP
C642:A6 2B     43 RDDHDR    LDX    SLOTZ
C644:C6 03     44          DEC    $03
C646:D0 0E C656 45          BNE    RDHDO
C648:BD 88 C0   46 FUGIT    LDA    $C088,X
C64B:BD CF C6   47 FUG1    LDA    MSG-$60,X
C64E:10 FE C64E 48 HANGING  BPL    HANGING
C650:9D 7B 07   49          STA    BOOTSCRN-$60,X
C653:E8        50          INX
C654:80 F5 C64B 51          BRA    FUG1
C656:08        52 RDHDO    PHP
C657:88        53 RETRY    DEY
C658:D0 04 C65E 54          BNE    RDHD1
C65A:F0 E5 C641 55          BEQ    RETRY1
C65C:80 DF C63D 56 EXTENT   BRA    EXTENT1
C65E:        57 * * * * *
C65E:        58 * The following code is sacred in it's *
C65E:        59 * present form. To change it would *
C65E:        60 * cause volcanos to erupt, the ground *

```



```

C65E:          61 * to shake, and ProDOS not to boot!      *
C65E:          62 * * * * * * * * * * * * * * * * * * *
C65E:BD 8C C0  63 RDHD1      LDA    $C08C,X
C661:10 FB C65E 64          BPL    RDHD1
C663:49 D5      65 ISMRK1     EOR    #$D5
C665:D0 F0 C657 66          BNE    RETRY
C667:BD 8C C0  67 RDHD2      LDA    $C08C,X
C66A:10 FB C667 68          BPL    RDHD2
C66C:C9 AA      69          CMP    #$AA
C66E:D0 F3 C663 70          BNE    ISMRK1
C670:EA        71          NOP
C671:BD 8C C0  72 RDHD3      LDA    $C08C,X
C674:10 FB C671 73          BPL    RDHD3
C676:C9 96      74          CMP    #$96
C678:F0 09 C683 75          BEQ    RDSECT
C67A:28        76          PLP
C67B:90 C2 C63F 77          BCC    RDADR
C67D:49 AD      78          EOR    #$AD
C67F:F0 25 C6A6 79          BEQ    RDATA
C681:D0 8C C63F 80          BNE    RDADR
C683:A0 03      81 RDSECT     LDY    #$03
C685:85 40      82 RDSEC1     STA    $40
C687:BD 8C C0  83 RDSEC2     LDA    $C08C,X
C68A:10 FB C687 84          BPL    RDSEC2
C68C:2A        85          ROL    A
C68D:85 3C      86          STA    BOOTTMP
C68F:BD 8C C0  87 RDSEC3     LDA    $C08C,X
C692:10 FB C68F 88          BPL    RDSEC3
C694:25 3C      89          AND    BOOTTMP
C696:88        90          DEY
C697:D0 EC C685 91          BNE    RDSEC1
C699:28        92          PLP
C69A:C5 3D      93          CMP    $3D
C69C:D0 A1 C63F 94          BNE    RDADR
C69E:A5 40      95          LDA    $40
C6A0:C5 41      96          CMP    $41
C6A2:D0 9B C63F 97 BADRD1     BNE    RDADR
C6A4:B0 9C C642 98          BCS    RDDHDR
C6A6:A0 56      99 RDATA      LDY    #$56
C6A8:B4 3C      100 RDATA0     STY    BOOTTMP
C6AA:BC 8C C0  101 RDATA1     LDY    $C08C,X
C6AD:10 FB C6AA 102          BPL    RDATA1
C6AF:59 D6 02   103          EOR    DNIBL-$80,Y
C6B2:A4 3C      104          LDY    BOOTTMP
C6B4:88        105          DEY
C6B5:99 00 03   106          STA    NBUF1,Y
C6B8:D0 EE C6AB 107          BNE    RDATA0
C6BA:84 3C      108 RDATA2     STY    BOOTTMP
C6BC:BC 8C C0  109 RDATA3     LDY    $C08C,X
C6BF:10 FB C6BC 110          BPL    RDATA3
C6C1:59 D6 02   111          EOR    DNIBL-$80,Y
C6C4:A4 3C      112          LDY    BOOTTMP
C6C6:91 26      113          STA    ($26),Y
C6C8:C8        114          INY
C6C9:D0 EF C6BA 115          BNE    RDATA2
C6CB:BC 8C C0  116 RDATA4     LDY    $C08C,X
C6CE:10 FB C6CB 117          BPL    RDATA4
C6D0:59 D6 02   118          EOR    DNIBL-$80,Y

```

```

C6D3:00 0D C6A2 119 BADREAD BNE BADRD1
C6D5:A0 00 120 LDY #$00
C6D7:A2 56 121 DENIBL LDX #$56
C6D9:CA 122 DENIB1 DEX
C6DA:30 FB C6D7 123 BMI DENIBL
C6DC:B1 26 124 LDA ($26),Y
C6DE:5E 00 03 125 LSR NBUF1,X
C6E1:2A 126 ROL A
C6E2:5E 00 03 127 LSR NBUF1,X
C6E5:2A 128 ROL A
C6E6:91 26 129 STA ($26),Y
C6E8:C8 130 INY
C6E9:D0 EE C6D9 131 BNE DENIB1
C6EB: 132 * * * * *
C6EB: 133 * Code beyond this point is not *
C6EB: 134 * sacred... It may be perverted *
C6EB: 135 * in any manner by any pervert. *
C6EB: 136 * * * * *
C6EB:E6 27 137 INC $27
C6ED:E6 3D 138 INC $3D
C6EF:A5 3D 139 LDA $3D
C6F1:CD 00 08 140 CMP $0800
C6F4:A6 4F 141 LDX BOOTDEV
C6F6:90 DB C6D3 142 BCC BADREAD
C6F8:4C 01 08 143 JMP $0801
C6FB:4C 0B C6 144 DODRV2 JMP DRV2ENT
C6FE: 0002 145 DS $C700-*,0
C700:FF 146 DFB $FF ;MAKE IT LOOK LIKE NOTHING IN SLOT
C701:A9 E0 147 DRV2BOOT LDA #$E0 ;FOR DEVICE #2
C703:A0 01 148 LDY #1 ;TO SELECT DRIVE 2
C705:A2 60 149 LDX #$60
C707:80 F2 C6FB 150 BRA DODRV2
C709:A2 03 151 MAKTBL LDX #$03
C70B:A0 00 152 LDY #0
C70D:86 3C 153 TBLLOOP STX BOOTTMP
C70F:8A 154 TXA
C710:0A 155 ASL A
C711:24 3C 156 BIT BOOTTMP
C713:F0 10 C725 157 BEQ NOPATRN
C715:05 3C 158 ORA BOOTTMP
C717:49 FF 159 EOR #$FF
C719:29 7E 160 AND #$7E
C71B:80 08 C725 161 TBLLOOP2 BCS NOPATRN
C71D:4A 162 LSR A
C71E:D0 FB C71B 163 BNE TBLLOOP2
C720:98 164 TYA
C721:9B 56 03 165 STA DNIBL,X
C724:C8 166 INY
C725:E8 167 NOPATRN INX
C726:10 E5 C70D 168 BPL TBLLOOP
C728:A9 08 169 LDA #$08
C72A:85 27 170 STA $27
C72C:A0 7F 171 LDY #$7F
C72E:60 172 RTS
C72F: C72F 173 MSG EQU *
C72F: 174 MSB ON
C72F:C3 E8 E5 E3 175 ASC 'Check Disk Drive.'
C740: 176 *

```

```

C740:      177 * The following code is Teri's memory and
C740:      178 * soft switch exercise program. The only
C740:      179 * purpose is exercise, not diagnostic
C740:      180 * functions. This code is activated on
C740:      181 * a system without a keyboard, or when
C740:      182 * both open and closed apple keys are
C740:      183 * pressed during the reset sequence.
C740:      184 *
C740:08 50 52      185 TBL1      DFB      $08,$50,$52      ;These are low order
C743:00 02 04      186      DFB      $00,$02,$04      ; addresses of $C0XX
C746:8B 8B E8      187      DFB      $88,$88,$88      ; that must be re-selected
C749:09 50 52      188 TBL2      DFB      $09,$50,$52      ; after each page write
C74C:00 03 05      189      DFB      $00,$03,$05      ; (especially $C0001)
C74F:83 83 E8      190      DFB      $83,$83,$E8
C752:      191 *
C752:64 00      192 XLOOP1    STZ      $00      ;Reset low address to 2
C754:E6 00      193      INC      $00      ;Hi addr assumed to = 0
C756:E6 00      194      INC      $00
C758:92 00      195 XPAGE     STA      ($00)      ;Write entire page with
C75A:9D 00 C0      196      STA      $C000,X      ; shifted data... BUT
C75D:6A      197      ROR      A      ; restore Z-page after
C75E:E6 00      198      INC      $00      ; write in case $C008-9
C760:D0 F6 C758 199      BNE      XPAGE      ; is current pointer
C762:18      200      CLC      ;Indicates regular pass
C763:98      201 XMODE     TYA      ;Get settings, each bit
C764:AD 08      202      LDY      #$08      ;Specifies main/alt set
C766:BE 40 C7      203 XRSET     LDX      TBL1,Y      ;Assume Main $C000 setting
C769:90 03 C76E 204      BCC      XRST1      ;Branch if Main setting
C76B:BE 49 C7      205      LDX      TBL2,Y      ;Else get Alternate index
C76E:9D 00 C0      206 XRST1    STA      $C000,X
C771:2A      207      ROL      A      ;Accumulator makes full
C772:88      208      DEY      ; circle
C773:10 F1 C766 209      BPL      XRSET
C775:A8      210      TAY      ;Preserve settings in Y
C776:80 DA C752 211      BCS      XLOOP1      ;Branch if new setting
C778:E6 01      212      INC      $01
C77A:D0 DC C758 213      BNE      XPAGE      ;Loop til all pages written
C77C:38      214 BANGER     SEC      ;Indicate new settings,
C77D:C8      215      INY      ; reset mem pointer after
C77E:80 E3 C763 216      BRA      XMODE      ; after new settings
C780:      0000 217      DS      $C780-*
C780:      24      INCLUDE MBASIC      ;Mouse BASIC routines @$C780

```

```

C780:      3 *****
C780:      4 *
C780:      5 * XMBASIC - Basic call to the mouse
C780:      6 *
C780:      7 *****
C780:      C780 8 xmbasic equ *
C780:5A      9 phy
C781:B0 1C C79F 10 bcs basicin ;Input?
C783:A0 C4      11 ldy #<mbasic ;Input from $C400?
C785:C4 39      12 cpy kswl
C787:D0 04 C78D 13 bne xmbout
C789:A4 38      14 ldy kswl
C78B:E0 12 C79F 15 beq basicin
C78D:DA      16 xmbout phx ;Save X too
C78E:48      17 pha
C78F:29 7F      18 and #$7F ;We don't care about high bit
C791:C9 02      19 cmp #2
C793:B0 06 C79B 20 bge mbbad ;Only 0,1 valid
C795:20 3D C4      21 jsr xsetmou
C798:20 6D C4      22 jsr xhome
C79B:68      23 mbbad pla
C79C:FA      24 plx
C79D:7A      25 ply
C79E:60      26 rts

C79F:      28 *****
C79F:      29 *
C79F:      30 * BASICIN - Input from basic
C79F:      31 *
C79F:      32 * Creates +XXXXX,+YYYYY,+SS
C79F:      33 * XXXXX = X position
C79F:      34 * YYYYY = Y position
C79F:      35 * SS = Status
C79F:      36 * - = Key pressed
C79F:      37 * 1 = Button pressed
C79F:      38 * 2 = Button just pressed
C79F:      39 * 3 = Button just released
C79F:      40 * 4 = Button not pressed
C79F:      41 *
C79F:      42 *****
C79F:      C79F 43 basicin equ *
C79F:91 28      44 sta (basl),y ;Fix flashing char
C7A1:A9 05      45 lda #>inent ;Fix input entry
C7A3:85 38      46 sta kswl
C7A5:AD 00 C0      47 lda kbd ;test the keyboard
C7A8:0A      48 asl A
C7A9:08      49 php ;Save kbd and int stat for later
C7AA:78      50 sei ;No interrupts while getting position
C7AB:20 95 C4      51 jsr xmread
C7AE:A0 05      52 ldy #5 ;Move X position into the buffer
C7B0:AE 7C 05      53 ldx mouxh
C7B3:AD 7C 04      54 lda mouxl
C7B6:20 9B C5      55 jsr hexodec ;Convert it
C7B9:A0 0C      56 ldy #12
C7BB:AE FC 05      57 ldx mouyh
C7BE:AD FC 04      58 lda mouyl

```

```

C7C1:20 9B C5      59      jsr    hextohex
C7C4:AD 7C 07      60      lda    moustat
C7C7:2A            61      rol    A
C7C8:2A            62      rol    A
C7C9:2A            63      rol    A
C7CA:29 03         64      and    #3
C7CC:49 03         65      eor    #3
C7CE:1A            66      inc    A
C7CF:28            67      plp
C7D0:A0 10         68      ldy    #16      ;Restore int & kbd status
C7D2:20 AC C5      69      jsr    hexdec2    ;X=0 from last div10
C7D5:7A            70      ply
C7D6:A2 11         71      ldx    #17      ;X = EOL
C7D8:A9 8D         72      lda    #$8D    ;Carriage return
C7DA:9D 00 02      73      putinbuf sta    inbuf,x
C7DD:60            74      rts

```

```

C7DE:              76      *****
C7DE:              77      *
C7DE:              78      * PADDLE patch
C7DE:              79      *
C7DE:              80      *****
C7DE:              C7DE 81      mpaddle equ    *
C7DE:AD FC 07      82      lda    moumode    ;Is the mouse active?
C7E1:C9 01         83      cmp    #01      ;Only transparent mode
C7E3:F0 06         C7EB 84      beq    pdon
C7E5:AD 70 C0      85      lda    vblclr    ;Fire the strobe
C7E8:4C 21 FB      86      jmp    $FB21
C7EB:              C7EB 87      pdon equ    *
C7EB:E0 01         88      cpx    #1      ;C=1 if X=1
C7ED:6A            89      ror    A      ;A=80 or 0
C7EE:A8            90      tay
C7EF:B9 7C 05      91      lda    mouh,y    ;Get high byte
C7F2:F0 02         C7F6 92      beq    pdok
C7F4:A9 FF         93      lda    #$FF
C7F6:19 7C 04      94      pdok ora    mouh,y
C7F9:A8            95      tay
C7FA:60            96      rts
C7FB:5D E8 C5      97      zznml eor    qtbl,x
C7FE:80 DA         C7DA 98      bra    putinbuf
C800:              99      ds     $C800-*,0
C800:              25      INCLUDE IRQBUF    ;Interrupt stuff @C800

```

```

C800:          3 *
C800:          4 * this is the main (only) IRQ handling routines
C800:          5 *
C800:4C E4 C1   6      jmp    plinit      ;Pascal 1.0 Initialization
C803:48        7 NEWIRQ    PHA          ;SAVE ACC ON STACK, NOT $45
C804:68        8          PLA          ;LEGAL BECAUSE IF IRQ, IRQ DISABLED.
C805:68        9          PLA          ;GET STATUS REGISTER
C806:48       10 IRQ1     PHA          ;
C807:D8       11        CLD          ;CLEAR DEC MODE, ELSE THINGS GET SCREWED.
C808:29 10     12        AND    #$10    ;SET CARRY TO INDICATE BRK
C80A:69 F0     13        ADC    #$F0    ;
C80C:8A       14        TXA          ;SAVE X IN A WHILE
C80D:BA       15        TSX          ; FUTZING WITH THE STACK
C80E:CA       16        DEX          ; RECOVER A-REG AT TOP...
C80F:9A       17        TXS          ;
C810:48       18        PHA          ;SAVE X ON STACK (ON TOP OF A)
C811:5A       19        PHY          ; AND Y ALSO
C812:AE 66 C0  20        LDX    MOUX1   ;Get mouse info
C815:AC 67 C0  21        LDY    MOUY1   ;
C818:AD 18 C0  22        LDA    RD80COL  ;TEST FOR 80-STORE WITH
C81B:2D 1C C0  23        AND    RDPAGE2  ; PAGE 2 TEXT.
C81E:29 80     24        AND    #$80    ; MAKE IT ZERO OR $80
C820:F0 05 C827 25        BEQ    IRQ2    ;
C822:BD 54 C0  26        STA    TXTPAGE1 ;
C825:A9 40     27        LDA    #$40    ;SET PAGE 2 RESET BIT.
C827:2C 13 C0  28 IRQ2    BIT    RDRAMRD ;
C82A:10 05 C831 29        BPL    IRQ3    ;BRANCH IF MAIN RAM READ
C82C:8D 02 C0  30        STA    RDMAINRAM ;ELSE, SWITCH IT IN
C82F:09 20     31        ORA    #$20    ; AND RECORD THE EVENT!
C831:2C 14 C0  32 IRQ3    BIT    RDRAMWRT ;DO THE SAME FOR RAM WRITE.
C834:10 05 C83B 33        BPL    IRQ4    ;
C836:8D 04 C0  34        STA    WRMAINRAM ;
C839:09 10     35        ORA    #$10    ;
C83B:B0 13 C850 36 IRQ4    BCS    IRQ5    ;BRANCH IF BREAK, NOT INTERRUPT
C83D:48       37        PHA          ;SAVE MACHINE STATES SO FAR...
C83E:20 D5 C4  38        JSR    MOUSEINT ;GO TEST THE MOUSE
C841:90 3F C882 39        BCC    IRQDONE ;BRANCH IF IT WAS THE MOUSE
C843:20 0D C9  40        JSR    ACTAINT  ;GO TEST ACIA AND KEYBOARD INTERRUPTS
C846:90 3A C882 41        BCC    IRQDONE ;BRANCH IF INTERRUPT SERVICED
C848:68       42        PLA          ;RESTORE STATES RECORDED SO FAR
C849:18       43        CLC          ;RESET BREAK/INTERRUPT INDICATOR
C84A:80 04 C850 44        bra    passkipl ;Skip around pascal 1.0 stuff
C84C:        0001 45        ds        $C84D-*, $00
C84D:4C EE C1  46        jmp    pload    ;
C850:        C850 47 passkipl equ    *
C850:2C 12 C0  48 IRQ5    BIT    RDLDRAM  ;DETERMINE IF LANGUAGE CARD ACTIVE
C853:10 0C C861 49        BPL    IRQ7    ;
C855:09 0C     50        ORA    #$C      ;SET TWO BITS SO RESTORED
C857:2C 11 C0  51        BIT    RDLCBNK2 ; LANGUAGE CARD IS WRITE ENABLED
C85A:10 02 C85E 52        BPL    IRQ6    ;BRANCH IF NOT PAGE 2 OF $D000
C85E:49 06     53        EOR    #$6      ;ENABLE READ FOR PAGE 2 ON EXIT
C85E:8D 81 C0  54 IRQ6    STA    ROMIN   ;
C861:2C 16 C0  55 IRQ7    BIT    RDALTZP ;LAST...AND VERY IMPORTANT!
C864:10 0D C873 56        BPL    IRQ8    ; UNLESS IT IS NOT ENABLED
C866:BA       57        TSX          ;SAVE CURRENT STACK POINTER
C867:8E 01 01  58        STX    $101    ;AT BOTTOM OF STACK
C86A:AE 00 01  59        LDX    $100    ;GET MAIN STACK POINTER
C86D:9A       60        TXS          ;

```



```

C86E:8D 08 C0      61      STA      SETSTDZP
C871:09 80      62      ORA      #$80
C873:80 2A      C89F 63      IRQ8      BCS      GOBREAK
C875:48      64      PHA
C876:A9 C8      65      LDA      #<IRQDONE
C878:48      66      PHA
C879:A9 82      67      LDA      #>IRQDONE      ;SAVE RETURN IRQ ADDR
C87B:48      68      PHA
C87C:A9 04      69      LDA      #4      ; SO WHEN INTERRUPT DOES RTI
C87E:48      70      PHA      ; IT RETURNS TO IRQDONE.
C87F:6C FE 03      71      JMP      ($3FE)      ;PROCESS EXTERNAL INTERRUPT

C882:68      73      IRQDONE      PLA      ;RECOVER MACHINE STATE
C883:10 07      C88C 74      BPL      IRQDNE1      ;BRANCH IF MAIN ZP WAS ACTIVE
C885:8D 09 C0      75      STA      SETALTZP
C888:AE 01 01      76      LDX      $101      ;RESTORE ALTERNATE STACK POINTER
C88B:9A      77      TXS
C88C:0A      78      IRQDNE1      ASL      A
C88D:AD 05      79      LDY      #$05
C88F:BE 89 C9      80      IRQDNE2      LDX      IROTBLE,Y
C892:88      81      DEY
C893:0A      82      ASL      A
C894:90 03      C899 83      BCC      IRQDNE3      ;BRANCH IF SWITCH IS OK.
C896:9D 00 C0      84      STA      $C000,X
C899:D0 F4      C88F 85      IRQDNE3      SNE      IRQDNE2      ;BRANCH IF MORE SWITCHES
C89B:7A      86      PLY
C89C:FA      87      PLX      ;RESTORE ALL REGISTERS
C89D:68      88      PLA
C89E:40      89      RTI      ;DO THE REAL RTI!
C89F:4C 47 FA      90      GOBREAK      JMP      NEWBRK      ;PASS THE BREAKER THROUGH

C8A2:      92      *****
C8A2:      93      *
C8A2:      94      * MOVEIRO - This routine transfers the roms interrupt vector into
C8A2:      95      * both language cards
C8A2:      96      *
C8A2:      97      *****
C8A2:      C8A2 98      moveirq      equ      *
C8A2:20 60 C3      99      JSR      SETROM      ;Read ROM and Write to RAM
C8A5:AD 16 C0      100     LDA      RDALTZP      ;Which language card?
C8A8:0A      101     ASL      A      ;C=1 if alternate card
C8A9:A0 01      102     LDY      #1      ;Move two bytes
C8AB:B9 FE FF      103     MIRQLP      LDA      IRQVECT,Y      ;Get byte from ROM
C8AE:8D 09 C0      104     STA      SETALTZP      ;Set alternate card
C8B1:99 FE FF      105     STA      IRQVECT,Y      ;Store it in the RAM card
C8B4:8D 08 C0      106     STA      SETSTDZP      ;Set main card
C8B7:99 FE FF      107     STA      IRQVECT,Y
C8BA:88      108     DEY
C8BB:10 EE      C8AB 109     BPL      MIRQLP      ;Go do the second byte
C8BD:90 03      C8C2 110     BCC      MIRQSTD      ;Is the card set right?
C8BF:8D 09 C0      111     STA      SETALTZP      ;No, it wasn't
C8C2:4C 54 C3      112     MIRQSTD      JMP      RESETLC      ;Clean up & go home

```

```

C8C5:          114 *   This is the serial input routine.  Carry
C8C5:          115 * flag set indicates that returned data is
C8C5:          116 * valid.
C8C5:          117 *
C8C5:          118 * GETBUF- Gets a byte from the buffer & updates pointers
C8C5:          119 * On entry Y=0 for Serial buffer Y=$80 for Keyboard buffer
C8C5:EC FF 04    120 XRDSEB   CPX   ACIABUF   ;is serial input buffered?
C8C8:D0 26 C8F0 121         BNE   XNOSBUF   ;(in english "NO SERIAL BUFFER")
C8CA:A0 00      122         LDY   #0           ;Y=0 for serial buffer
C8CC:          123 GETBUF   EQU   *
C8CC:89 7F 06    124         LDA   TRSER,Y       ;Test for data in buffer
C8CF:D9 7F 05    125         CMP   TWSER,Y       ;If = then no data
C8D2:F0 24 C8F8 126         BEQ   GBEMPTY
C8D4:48          127         PHA           ;Save current value
C8D5:1A          128         INC   A           ;Update the pointer
C8D6:89 7F      129         BIT   #$7F       ;Overflow
C8D8:D0 01 C8D8 130         BNE   GBNOOVR
C8DA:98          131         TYA
C8DB:99 7F 06    132 GBNOOVR STA   TRSER,Y       ;Store the updated pointer
C8DE:7A          133         PLY           ;Get the old value of the pointer
C8DF:AD 13 C0    134         LDA   RDRAMRD      ;Are we in main ram
C8E2:0A          135         ASL   A           ;C=1 for Aux ram
C8E3:8D 03 C0    136         STA   RDCARDRAM   ;Force Aux ram
C8E6:B9 00 08    137         LDA   THBUF,Y       ;Get byte from buffer
C8E9:B0 14 C8FF 138         BCS   XRDSNO      ;Branch if we were in aux bank
C8EB:8D 02 C0    139         STA   RDMAINRAM   ;Set back to main
C8EE:38          140         SEC
C8EF:60          141         RTS           ;Note C=1
C8F0:          142 *
C8F0:BC 85 C8    143 XNOSBUF LDY   DEVNO,X       ;Get index to ACIA
C8F3:E9 F9 BF    144         LDA   SSTAT,Y       ;Test ACIA directly for data
C8F6:29 08      145         AND   #$8
C8F8:18          146 GBEMPTY CLC           ;indicate no data
C8F9:F0 04 C8FF 147         BEQ   XRDSNO      ;Branch if no data!
C8FB:B9 F8 BF    148         LDA   SDATA,Y       ;get serial input
C8FE:38          149 notacia SEC           ;indicate valid data returned.
C8FF:          150 acdone  equ   *
C8FF:60          151 XRDSNO   RTS

```

```

C900:      153 *   This routine will determine if the source of
C900:      154 *   is either of the built in ACIAs.  If neither port
C900:      155 *   generated the interrupt, or the interrupt was due
C900:      156 *   to a transmit buffer empty, protocol converter, or
C900:      157 *   'unbuffered' receiver full, the carry is set indi-
C900:      158 *   cating an externally serviced interrupt.
C900:      159 *   If the interrupt source was keyboard, 'buffered'
C900:      160 *   serial input, or the DCD, the interrupt is serviced
C900:      161 *   and the carry is cleared indicating interrupt was
C900:      162 *   serviced. (DCD handshake replaces CTS.)
C900:      163 *   Location "ACIABUF" specifies which (if either) re-
C900:      164 *   ceiver data is buffered.  For port 1 it must contain
C900:      165 *   $C1, for port 2 a $C2.  Any other values are cause
C900:      166 *   interrupts to pass to external (RAM based) routines.
C900:      167 *   Location "TYPHED" specifies whether Keyboard in-
C900:      168 *   put should be buffered, ignored, or processed by
C900:      169 *   RAM based routines.  If bit 7=1 and bit 6=0, key-
C900:      170 *   board data is placed in the type-ahead buffer.  If
C900:      171 *   bit 6 is set the interrupt is cleared, but must
C900:      172 *   be recognized and serviced by a RAM routine.  If
C900:      173 *   both bits = 0, the interrupt is serviced, but the
C900:      174 *   keyboard data is ignored.
C900:      175 *   While using type-ahead, Open-Apple CTRL-X will
C900:      176 *   flush the buffer.  No other code is recognized.
C900:      177 *   IF the source was an ACIA that has the transmit
C900:      178 *   interrupt enabled, the original value of the ACIAs
C900:      179 *   status registers is preserved.  Automatic serial input
C900:      180 *   buffering is not serviced from a port so configured.
C900:      181 *   Interrupts originating from the protocol converter or
C900:      182 *   keyboard (RAM serviced) do not inhibit serial buffering
C900:      183 *   and are passed thru.  The RAM service routine can rec-
C900:      184 *   ognize the interrupt source by a 1 state in bit 6 of
C900:      185 *   the ACIAs status register.  The RAM service routine must
C900:      186 *   cause the clearing of DSR (bit 6) AND make a second ac-
C900:      187 *   cess to the status register before returning.
C900:      188 *
C900:      189 *
C900:      190 aciainc equ *
C900:A2 C2      191     ldx    #<comslot      ;Test port 2 first
C902:20 08 C9      192     jsr    aciatst      ;Check for interrupt
C905:90 F8 C8FF      193     bcc    acdone      ;Return if interrupt done
C907:CA          194     dex     ;Try port 1
C908:BC 85 C8      195 aciatst ldy    devno,x      ;Get index for acia
C908:A9 04          196     lda     #$4        ;If xmit ints enabled pass to user
C90D:59 FA BF      197     eor    scomd,y      ;Check if D<3>, D<2> = 01
C910:29 0C          198     and    #$0C      ;
C912:F0 FA C8FE      199     beq    notacia      ;User better take it!
C914:89 F9 BF      200     lda     sstat,y      ;Get status
C917:9D 38 04      201     sta     astat,x      ;Save it away
C91A:10 E2 C8FE      202     bpl    notacia      ;No interrupt
C91C:E0 C2          203 aiatst2 cpx    #<comslot      ;C=1 if com port
C91E:B0 02 C922      204     bcs    alport2      ;Invert DSR if port1
C920:49 40          205     eor     #$40      ;
C922:3C 38 05      206 alport2 bit    extint,x      ;Is DSR enabled?
C925:70 26 C94D      207     bvs    alpass      ;Yes, user wants it
C927:10 22 C94B      208     bpl    aieatit      ;No, eat it
C929:90 20 C94B      209     bcc    aieatit      ;Yes but I don't want it For port 1
C92B:89 40          210     bit     #$40      ;Is DSR 1?

```

```

C92D:F0 1E C94D 211      beq  aipass      ;If not, skip it
C92F:      212 * It's a keyboard interrupt
C92F:AD 00 C0 213      lda  kbd          ;Get the key
C932:A0 80 214      ldy  #$80
C934:20 67 C9 215      jsr  putbuf      ;Put it in the buffer
C937:C9 98 216      cmp  #$98      ;Is it a ^x?
C939:D0 08 C943 217     bne  ainofish
C93B:AD 62 C0 218      lda  butnl      ;And the closed apple?
C93E:10 03 C943 219     bpl  ainofish
C940:20 1B CB 220      jsr  flush      ;Flush the buffer
C943:AD 10 C0 221     ainofish lda  kbdstrb    ;Clear the keyboard
C946:      222 * $A0 $B0 table needed by serial firmware
C946:      00C1 223     sltdmy equ  <ser slot
C946:      C885 224     devno equ  *--sltdmy
C946:A0 B0 225      ldy  #$B0      ;Restore y
C948:B9 F9 BF 226      lda  sstat,y    ;Read status to clear int
C94B:29 BF 227      and  #$BF      ;Clear the DSR bit
C94D:0A 228     aipass  sll  A      ;Shift DSR into C
C94E:0A 229      sll  A
C94F:29 20 230      and  #$20      ;Is the receiver full?
C951:F0 35 C988 231     beq  aciadone    ;If not, we're done
C953:B9 FA BF 232      lda  scomd,y    ;Are receive interrupts enabled?
C956:49 01 233      eor  #1      ;Check for D<1>,D<0> = 01
C958:29 03 234      and  #3
C95A:D0 2C C988 235     bne  aciadone    ;If not, were done
C95C:8A 236      txa
C95D:4D FF 04 237     eor  aciabuf    ;Is this acia buffered?
C960:D0 9C C8FE 238     bne  notacia    ;The user better handle it!
C962:B9 F8 BF 239      lda  sdata,y    ;It's mine
C965:A0 00 240      ldy  #0
C967:      C967 241     putbuf equ  *
C967:DA 242      phx
C968:48 243      pha
C969:B9 7F 05 244      lda  twser,y    ;Get buffer pointer
C96C:AA 245      tax      ;Save it for later
C96D:1A 246      inc  A      ;Bump it to next free byte
C96E:89 7F 247      bit  #$7F      ;Overflow?
C970:D0 01 C973 248     bne  pbok
C972:98 249      tya      ;Wrap pointer
C973:08 250     pbok  php      ;Save DSR status
C974:D9 7F 06 251     cmp  trser,y    ;Buffer full?
C977:F0 03 C97C 252     beq  pbfull
C979:99 7F 05 253     sta  twser,y    ;Save the new pointer
C97C:28 254     pbfull plp
C97D:68 255      pla      ;Get the data
C97E:8D 05 C0 256     sta  wrcardram    ;It goes to aux ram
C981:9D 00 08 257     sta  thbuf,x
C984:8D 04 C0 258     sta  wrmainram
C987:FA 259      plx
C988:60 260     aciadone rts

C989:83 8B 8B 262     IRQTBLE DFB  >LCBANK2,>LCBANK1,>LCBANK1
C98C:05 03 55 263     DFB  >WRCARDRAM,>RDCARDRAM,TXTPAGE2

```

```

C98F:      266 *   The following two routines are for reading key-
C98F:      267 * board and serial input from buffers or directly.
C98F:      268 *   Type-ahead buffering only occurs for non auto-
C98F:      269 * repeat keypresses. When a key is pressed for
C98F:      270 * auto-repeat the buffer is first emptied, then the
C98F:      271 * repeated characters are returned.
C98F:      272 *   The minus flag is used to indicate if a keystroke
C98F:      273 * is being returned.
C98F:      274 *

C98F:20 AD C9      276 XRDKBD      JSR   XBITKBD      ;is keyboard input ready?
C992:10 14 C9A8    277          BPL   XNOKEY      ;Branch if not.
C994:90 0A C9AD    278          BCC   XRKBD1      ;Branch if direct KBD input.
C996:5A          279          PHY          ;Save Y
C997:A0 80        280          LDY   #$80      ;Y=$80 for keyboard buffer
C999:20 CC C8      281          JSR   GETBUF      ;Get data from buffer
C99C:7A          282          PLY          ;
C99D:09 00        283          ORA   #0       ;Set minus flag
C99F:60          284          RTS

C9A0:AD 00 C0      286 XRKBD1      LDA   KBD       ;test keyboard directly
C9A3:10 EA C98F    287          BPL   XRDKBD      ;loop if buffered since test.
C9A5:8D 10 C0      288          STA   KBDSTRB    ;Clear keyboard strobe.
C9A8:60          289 XNOKEY      RTS       ;Minus flag indicates valid character

C9A9:      0001    291          ds    $C9AA-*, $00
C9AA:4C F6 C1      292          jmp   plwrire    ;Pascal 1.0 entry point

C9AD:2C FA 05      294 XBITKBD      BIT   TYPHED    ;This routine replaces "BIT KBD" instrucs
C9BD:10 10 C9C2    295          BPL   XBKB2      ; so as to function with type-ahead.
C9B2:38          296          SEC          ;anticipate data in buffer is ready
C9B3:08          297          PHP          ;save carry and minus flags
C9B4:48          298          PHA          ;preserve accumulator
C9B5:AD FF 06      299          LDA   TRKEY
C9B8:CD FF 05      300          CMP   TWKEY      ;is there data to be read?
C9BB:F0 03 C9C0    301          BEQ   XBKB1      ;branch if type-ahead buffer empty
C9BD:68          302          PLA
C9BE:28          303          PLP
C9BF:60          304          RTS       ;Carry and minus flag already set.
C9C0:      305 *
C9C0:58          306 XBKB1      PLA
C9C1:28          307          PLP       ;restore ACC and Status
C9C2:2C 00 C0      308 XBKB2      BIT   KBD       ;test KBD Directly
C9C5:18          309          CLC       ;indicate direct test
C9C6:60          310          RTS
C9C7:      26      INCLUDE COMMAND ;Serial firmware command processor

```



```

C9C7:          3      MSB      OFF
C9C7:          C9C7  4 cmdtable equ  *
C9C7:66        5      dfb      >cmdi-1
C9C8:66        6      dfb      >cmdk-1
C9C9:66        7      dfb      >cmdl-1
C9CA:5C        8      dfb      >cmdn-1
C9CB:5C        9      dfb      >cmdcr-1
C9CC:7C       10      dfb      >cmdb-1
C9CD:78       11      dfb      >cmdd-1
C9CE:77       12      dfb      >cmdp-1
C9CF:C3       13      dfb      >cmdq-1
C9D0:B4       14      dfb      >cmdr-1
C9D1:98       15      dfb      >cmds-1
C9D2:C5       16      dfb      >cmdt-1
C9D3:54       17      dfb      >cmdz-1
C9D4:7F BF BF 7F 18 mask1  dfb      $7F,$BF,$BF,$7F,$FF
C9D9:80 00 40 00 19 mask2  dfb      $80,$00,$40,$00,$00
C9DE:          C9DE  20 cmdlist equ  *
C9DE:49 4B 4C 4E 21      asc      "IKLN"
C9E2:0D       22      dfb      $0D          ;Carriage return
C9E3:42 44 50 51 23      asc      "BDPQRSTZ"
C9E8:48       24 command pha          ;Check for command to firmware
C9EC:3C B8 03   25      bit      sermode,x    ;Already in command?
C9EF:30 1B      CA0C 26      bmi      incmd      ;If so, go do it
C9F1:BC 38 06   27      idy      eschar,x      ;If eschar = 0 ignore commands
C9F4:F0 13      CA09 28      beq      nocmd
C9F6:5D 38 06   29      eor      eschar,x      ;Is it the command char?
C9F9:0A       30      asl      A              ;Ignore high bit
C9FA:D0 0D      CA09 31      bne      nocmd
C9FC:AC FB 07   32      ldy      cursor        ;Save the cursor
C9FF:8C 79 06   33      sty      oldcur
CA02:A0 BF      34      ldy      #cmdcur        ;Set command cursor
CA04:8C FB 07   35      sty      cursor
CA07:80 2D      CA36 36      bra      cominit
CA09:38       37 nocmd  sec          ;Mark char not handled
CA0A:68       38      pla          ;Restore char
CA0B:60       39      rts
CA0C:          CA0C  40 incmd  equ  *          ;Command mode
CA0C:BC 85 C8   41      ldy      devno,x      ;Get index for ACIA
CA0F:29 5F      42      and      #$5F          ;Ignore hi bit: just upshift lowercase
CA11:DA       43      phx
CA12:A2 0C      44      ldx      #12          ;Check 13 commands
CA14:DD DE C9   45 cmdloop cmp      cmdlist,x
CA17:F0 34      CA4D 46      beq      cmdfound    ;Right char?
CA19:CA       47      dex
CA1A:10 F8      CA14 48      bpl      cmdloop
CA1C:FA       49      plx          ;We didn't find it
CA1D:68       50      pla
CA1E:48       51      pha
CA1F:29 7F      52      and      #$7F          ;if char is cntl char
CA21:C9 20      53      cmp      #$20          ;it can be the new cmd char
CA23:B0 03      CA28 54      bcs      ckdig      ;branch if not cntl character
CA25:9D 38 06   55 cmdz2  sta      eschar,x    ;Save cmd char, drop thru ckdig to cdone
CA28:49 30      56 ckdig  eor      #$30          ;Is it a number?
CA2A:C9 0A      57      cmp      #$0A
CA2C:B0 0E      CA3C 58      bcs      cdone      ;If so, branch
CA2E:A0 0A      59      ldy      #10          ;A = A + 10 * current number
CA30:6D 7F 04   60 digloop adc      number    ;C=0 on first entry

```



CA33:88		61	dey		
CA34:D0 FA CA30		62	bne	digloop	
CA36:8D 7F 04		63	cominc	sta	number
CA39:38		64	sec		;Mark in command mode
CA3A:80 07 CA43		65	bra	cmset	
CA3C:18		66	cdone	clc	;Out of command mode
CA3D:AD 79 06		67	lda	oldcur	;Restore the cursor
CA40:8D FB 07		68	sta	cursor	
CA43:08		69	cmset	php	
CA44:1E B8 03		70	asl	sermode,x	;set command mode according to carry
CA47:28		71	plp		
CA48:7E B8 03		72	ror	sermode,x	;leaves carry clear
CA4B:68		73	pla		;character handled
CA4C:60		74	rts		;because carry clear...
CA4D:A9 CA		76	cmfound	lda	#<cmdcr
CA4F:48		77	pha		;do JMP via RTS
CA50:BD C7 C9		78	lda	cmdtable,x	
CA53:48		79	pha		
CA54:60		80	rts		;Go to it
CA55:FA		82	cmdz	plx	;Zero escape character
CA56:9E B8 04		83	stz	pwdth,x	;And the width
CA59:A9 00		84	lda	#0	
CA5B:80 C8 CA25		85	brs	cmdz2	
CA5D:	CA5D	87	cmdcr	equ	*
CA5D:	CA5D	88	cmdn	equ	*
CA5D:7A		89	ply		
CA5E:AD 7F 04		90	lda	number	;Get number inputted
CA61:F0 05 CA68		91	beq	cmd12	;Don't change printer width if 0
CA63:99 B8 04		92	sta	pwdth,y	;Update printer width
CA66:F0		93	dfb	\$F0	;BEQ opcode to skip next byte
CA67:	CA67	94	cmd1	equ	*
CA67:	CA67	95	cmdk	equ	*
CA67:	CA67	96	cmdl	equ	*
CA67:7A		97	ply		
CA68:B9 B8 06		98	cmd12	lda	flags,y
CA6B:3D D4 C9		99	and	mask1,x	;Mask off bit we'll change
CA6E:1D D9 C9		100	ora	mask2,x	;Change it
CA71:99 B8 06		101	sta	flags,y	;Back it goes
CA74:98		102	tya		;Put slot back in x
CA75:AA		103	tax		
CA76:80 C4 CA3C		104	cdone2	bra	cdone
					;Good bye
CA78:88		106	cmdp	dey	;Make y point to command reg
CA79:A9 1F		107	cmdd	lda	#\$1F
CA7B:38		108	sec		;Mask off high three bits
CA7C:90		109	dfb	\$90	;C=1 means high 3 bits
CA7D:A9 F0		110	cmdb	lda	#\$F0
CA7F:18		111	clc		;BCC opcode to skip next byte
CA80:39 FB BF		112	and	scntl,y	;Mask off lower 4 bits F0 = BNE
					;F0 will skip this if cmdp or cmdd
					;Mask off bits being changed

CA83:8D F8 06	113	sta	temp	;Save it
CA86:FA	114	plx		
CA87:AD 7F 04	115	lda	number	;Get inputed number
CA8A:29 0F	116	and	#\$0F	;Only lower nibble valid
CA8C:90 05 CA93	117	bcc	noshift	;If C=1 shift to upper 3 bits
CA8E:0A	118	asl	A	
CA8F:0A	119	asl	A	
CA90:0A	120	asl	A	
CA91:0A	121	asl	A	
CA92:0A	122	asl	A	
CA93:0D F8 06	123	noshift	ora	temp
CA96:C8	124	iny		;Get the rest of the bits
CA97:80 17 CAB0	125	bra	cmdp2	;Put them in the ACIA
				;increment puts em away where they go.
CA99:B9 FA BF	127	cmds	lda	scomd,y
CA9C:48	128		pha	;Transmit a break
CA9D:09 0C	129		ora	;\$0C
CA9F:99 FA BF	130		sta	scomd,y
CAA2:A9 E9	131		lda	#233
CAA4:A2 53	132	mwait	ldx	#83
CAA6:48	133	msloop	pha	;For 233 ms
CAA7:68	134		pla	;Wait 1 ms
CAA8:CA	135		dex	;((12*82)+11)+2+3=1000us
CAA9:D0 FB CAA6	136		bne	msloop
CAAB:3A	137		dec	a
CAAC:D0 F6 CAA4	138		bne	mwait
CAAE:68	139		pla	
CAAF:FA	140		plx	
CAB0:99 FA BF	141	cmdp2	sta	scomd,y
CAB3:80 C1 CA76	142		bra	cdone2
CAB5:99 F9 BF	144	cmdr	sta	sstat,y
CAB8:AD 78 06	145		lda	vfactv
CABB:0A	146		asl	A
CABC:20 23 CE	147		jsr	sethooks
CABF:90 03 CAC4	148		bcc	cmdq
CAC1:20 4D CE	149		jsr	zzquit
CAC4:18	150	cmdq	clc	;Reset the hooks
CAC5:80	151		dfb	;\$B0
CAC6:38	152	cmdt	sec	;Quit terminal mode
CAC7:FA	153		plx	;BCS to skip next byte
CAC8:20 CD CA	154		jsr	setterm
CACB:80 A9 CA76	155		bra	cdone2
CACD:BD B8 03	156	setterm	lda	sermode,x
CAD0:89 40	157		bit	;\$40
CAD2:90 12 CAE6	158		bcc	stclr
CAD4:D0 20 CAF6	159		bne	stwasok
CAD6:E4 39	160		cpx	kswh
CAD8:D0 47 CB21	161		bne	strts
CADA:09 40	162		ora	;\$40
CADC:AC 79 06	163		ldy	oldcur
CADF:8C 7A 06	164		sty	oldcur2
CAE2:A0 DF	165		ldy	#termcur
CAE4:80 07 CAED	166		bra	stset
				;Get terminal mode status
				;Z=1 if not in terminal mode
				;Branch if clearing terminal mode
				;Was already set
				;Are we in the input hooks
				;Leaves C=1 if =
				;Set term mode bit
				;Save what was in oldcur
				;Get new cursor value

```

CAE6:F0 0E CAF6 167 stclr    beq    stwasok    ;Branch if already clear
CAE8:29 BF          168      and    #$BF        ;Clear the bit
CAEA:AC 7A 06      169      ldy    oldcur2     ;Restore the cursor
CAED:9D B8 03      170 stset    sta    sermode,x
CAFO:8C 79 06      171      sty    oldcur        ;Save cursor to be restored after command
CAF3:8C FB 07      172      sty    cursor
CAF6:BC 85 C8      173 stwasok ldy    devno,x
CAF9:58          174      cli                    ;want to leave with interrupts active
CAFA:08          175      php
CAF8:78          176      sei                    ;but off while we twiddle bits
CAFC:B9 FA BF      177      lda    scomd,y
CAFF:09 02          178      ora    #$2          ;disable receiver interrupts if
CB01:90 02 CB05     179      bcc    cmdt2        ; not in terminal mode
CB03:29 FD          180      and    #$FD        ;enable when in terminal mode
CB05:99 FA BF      181 cmdt2   sta    scomd,y
CB08:A9 00          182      lda    #0
CB0A:6A          183      ror    a                ;set kbd interrupts according to t-mode
CB0B:8D FA 05      184      sta    typhed
CB0E:10 07 CB17     185      bpl    cmdt3        ;branch if leaving terminal mode
CB10:9C 7F 05      186      stz    twser        ; and ser buf...
CB13:9C 7F 06      187      stz    trser
CB16:8A          188      txa                    ;use x to enable serial buffering
CB17:8D FF 04      189 cmdt3   sta    aciabuf
CB1A:28          190      plp                    ;restore carry, enable interrupts.
CB1B:8E FF 05      191 flush   stx    twkey        ;Flush the type ahead buffer
CB1E:8E FF 06      192      stx    trkey
CB21:60          193 strts   rts
CB22:          0002 194      ds     $CB24-*, $00
CB24:E8          195 zzn2    inx
CB25:4C FB C7      196      jmp    zzn1
CB28:9E 0B 40 50   197 comtbl dfb    $9E,$0B,$40,$50,$16,$0B,$01,$00
CB30:          27      INCLUDE SCROLLING    ;More Video stuff @ $CB30

```

```

CB30:      3 *
CB30:      4 * SCROLLIT scrolls the screen either up or down, depending
CB30:      5 * on the value of X. It scrolls within windows with even
CB30:      6 * or odd edges for both 40 and 80 columns. It can scroll
CB30:      7 * windows down to 1 character wide.
CB30:      8 *
CB30:DA     9 SCROLLDN PHX          ;save X
CB31:A2 00   10         LDX #0      ;direction = down
CB33:80 03   11         BRA SCROLLIT ;do scroll
CB35:      12 *
CB35:DA     13 SCROLLUP PHX          ;save X
CB36:A2 01   14         LDX #1      ;direction = up
CB38:A4 21   15 SCROLLIT LDY WNDWDTH ;get width of screen window
CB3A:2C 1F C0 16         BIT RD80VID ;in 40 or 80 columns?
CB3D:10 18   17         BPL GETST    ;=>40, determine starting line
CB3F:8D 01 C0 18         STA SET80COL ;make sure this is enabled
CB42:98      19         TYA          ;get WNDWDTH for test
CB43:4A      20         LSR A        ;divide by 2 for 80 column index
CB44:A8      21         TAY          ;and save
CB45:A5 20   22         LDA WNDLFT   ;test oddity of right edge
CB47:4A      23         LSR A        ;by rotating low bit into carry
CB48:B8      24         CLV          ;V=0 if left edge even
CB49:90 03   25         BCC CHKRT    ;=>check right edge
CB4B:2C C1 CB 26         BIT SEV1     ;V=1 if left edge odd
CB4E:2A      27 CHKRT   ROL A        ;restore WNDLFT
CB4F:45 21   28         EOR WNDWDTH  ;get oddity of right edge
CB51:4A      29         LSR A        ;C=1 if right edge even
CB52:70 03   30         BVS GETST    ;if odd left, don't DEY
CB54:B0 01   31         BCS GETST    ;if even right, don't DEY
CB56:88      32         DEY          ;if right edge odd, need one less
CB57:8C F8 05 33 GETST  STY TEMPY    ;save window width
CB5A:AD 1F C0 34         LDA RD80VID  ;N=1 if 80 columns
CB5D:08      35         PHP          ;save N,Z,V
CB5E:A5 22   36         LDA WNDTOP   ;assume scroll from top
CB60:E0 00   37         CPX #0       ;up or down?
CB62:D0 03   38         BNE SETDBAS  ;=>up
CB64:A5 23   39         LDA WNDBTM   ;down, start scrolling at bottom
CB66:3A      40         DEC A        ;really need one less
CB67:      41 *
CB67:8D 78 05 42 SETDBAS STA TEMPA    ;save current line
CB6A:20 24 FC 43         JSR VTBZ     ;calculate base with window width
CB6D:      44 *
CB6D:A5 28   45 SCRLIN  LDA BASL     ;current line is destination
CB6F:85 2A   46         STA BAS2L
CB71:A5 29   47         LDA BASH
CB73:85 2B   48         STA BAS2H
CB75:      49 *
CB75:AD 78 05 50         LDA TEMPA    ;get current line
CB78:E0 00   51         CPX #0       ;going up?
CB7A:D0 07   52         BNE SETUP2   ;=>up, inc current line
CB7C:C5 22   53         CMP WNDTOP   ;down. Reached top yet?
CB7E:F0 39   54         BEQ SCRL3    ;yes! clear top line, exit
CB80:3A      55         DEC A        ;no, go up a line
CB81:80 05   56         BRA SETSRC   ;set source for scroll
CB83:1A      57 SETUP2  INC A        ;up, inc current line
CB84:C5 23   58         CMP WNDBTM   ;at bottom yet?
CB86:B0 31   59         BCS SCRL3    ;yes! clear bottom line, exit
CB88:      60 *

```

CB88:8D 78 05	61	SETSRC	STA	TEMPA	;save new current line
CB8B:20 24 FC	62		JSR	VTABZ	;get base for new current line
CB8E:AC F8 05	63		LDY	TEMPY	;get width for scroll
CB91:28	64		PLP		;get status for scroll
CB92:08	65		PHP		;N=1 if 80 columns
CB93:10 1F CBB4	66		BPL	SKPRT	;=>only do 40 columns
CB95:AD 55 C0	67		LDA	TXTPAGE2	;scroll aux page first (even bytes)
CB98:98	68		TYA		;test Y
CB99:F0 07 CBA2	69		BEQ	SCRLEFT	;if Y=0, only scroll one byte
CB9B:B1 28	70	SCRLEVEN	LDA	(BASL),Y	
CB9D:91 2A	71		STA	(BAS2L),Y	
CB9F:88	72		DEY		
CBA0:D0 F9 CB9B	73		BNE	SCRLEVEN	;do all but last even byte
CBA2:70 04 CBA8	74	SCRLEFT	BVS	SKPLFT	;odd left edge, skip this byte
CBA4:B1 28	75		LDA	(BASL),Y	
CBA6:91 2A	76		STA	(BAS2L),Y	
CBA8:AD 54 C0	77	SKPLFT	LDA	TXTPAGE1	;now do main page (odd bytes)
CBA8:AC F8 05	78		LDY	TEMPY	;restore width
CBAE:B0 04 CBB4	79		BCS	SKPRT	;even right edge, skip this byte
CBBC:B1 28	80	SCRLODD	LDA	(BASL),Y	
CBBC:91 2A	81		STA	(BAS2L),Y	
CBB4:88	82	SKPRT	DEY		
CBB5:10 F9 CBB0	83		BPL	SCRLODD	
CBB7:80 B4 CBB0	84		BRA	SCRLIN	;scroll next line
CBB9:	85	*			
CBB9:20 A0 FC	86	SCRCL3	JSR	CLRLIN	;clear current line
CBBC:20 22 FC	87		JSR	VTAB	;restore original cursor line
CBBF:28	88		PLP		;pull status off stack
CBC0:FA	89		PLX		;restore X
CBC1:60	90	SEV1	RTS		;done!!!

```

CBC2:          92 *
CBC2:          93 * DOCLR is called by CLREOL. It decides whether
CBC2:          94 * to do a (quick) 40 or 80 column clear to end of line.
CBC2:          95 *
CBC2:2C 1F C0 96 DOCLR      BIT   RD8OVID      ;40 or 80 column clear?
CBC5:30 13 CBDA 97          BMI   CLR80        ;=>clear 80 columns
CBC7:91 28      98 CLR40      STA   (BASL),Y
CBC9:C8        99          INY
CBCA:C4 21      100         CPY   WNDWDTH
CBCC:90 F9 CBC7 101         BCC   CLR40
CBCE:60        102         RTS
CBCF:          103 *
CBCF:DA        104 CLRHALF   PHX              ;clear right half of screen
CBDO:A2 D8      105         LDX   #5D8        ;for SCR48
CBD2:A0 14      106         LDY   #20
CBD4:A5 32      107         LDA   INVFLG
CBD6:29 A0      108         AND   #5A0
CBD8:80 17 CBF1 109         BRA   CLR2        ;=>jump into middle
CBDA:          110 *
CBDA:DA        111 CLR80     PHX              ;preserve X
CBDB:48        112         PHA              ;and blank
CBDC:98        113         TYA              ;get count for CH
CBDD:48        114         PHA              ;save for left edge check
CBDE:38        115         SEC              ;count=WNDWDTH-Y-1
CBDF:F5 21      116         SBC   WNDWDTH
CBE1:AA        117         TAX              ;save CH counter
CBE2:98        118         TYA              ;div CH by 2 for half pages
CBE3:4A        119         LSR   A
CBE4:A8        120         TAY
CBE5:68        121         PLA              ;restore original CH
CBE6:45 20      122         EOR   WNDLFT      ;get starting page
CBE8:6A        123         ROR   A
CBE9:80 03 CBEE 124         BCS   CLRO
CBEB:10 01 CBEE 125         BPL   CLRO
CBED:C8        126         INY              ;iff WNDLFT odd, starting byte odd
CBEE:68        127 CLRO     PLA              ;get blankity blank
CBEF:80 0B CBFC 128         BCS   CLR1        ;starting page is 1 (default)
CBF1:2C 55 C0 129 CLR2      BIT   TXTPAGE2    ;else do page 2
CBF4:91 28      130         STA   (BASL),Y
CBF6:2C 54 C0 131         BIT   TXTPAGE1      ;now do page 1
CBF9:E8        132         INX
CBFA:F0 06 CC02 133         BEQ   CLR3        ;all done
CBFC:91 28      134 CLR1     STA   (BASL),Y
CBFE:C8        135         INY              ;forward 2 columns
CBFF:E8        136         INX              ;next CH
CC00:D0 EF CBF1 137         BNE   CLR2        ;not done yet
CC02:FA        138 CLR3     PLX              ;restore X
CC03:60        139         RTS              ;and exit
CC04:          140 *
CC04:9C FA 05 141 CLRPORT   STZ   TYPHED      ;disable typeahead
CC07:9C F9 05 142         STZ   EXTINT2      ;and external interrupts
CC0A:60        143         RTS

```



```

CC0B:      145 *
CC0B:      146 * PASINVERT is used by Pascal to display the cursor. Pascal
CC0B:      147 * normally leaves the cursor on the screen at all times. It
CC0B:      148 * is fleetingly removed while a character is displayed, then
CC0B:      149 * promptly redisplayed. CTL-F and CTL-E, respectively,
CC0B:      150 * disable and enable display of the cursor when printed using
CC0B:      151 * the Pascal 1.1 entry point (PWRITE). Screen I/O is
CC0B:      152 * significantly faster when the cursor is disabled. This
CC0B:      153 * feature is supported by Pascal 1.2 and later.
CC0B:      154 *
CC0B:AD FB 04 155 PASINVERT LDA VMODE ;Called by pascal to
CC0E:29 10 156 AND #M.CURSOR ;display cursor
CC10:D0 0A CC1C 157 BNE INVX ;=>cursor off, don't invert
CC12: CC12 158 INVERT EQU *
CC12:20 1D CC 159 JSR PICKY ;load Y and get char
CC15:48 160 PHA
CC16:49 80 161 EOR #$80 ;FLIP INVERSE/NORMAL
CC1B:20 B3 C3 162 JSR STORY ;stuff onto screen
CC1B:68 163 PLA ;for RDCHAR
CC1C:60 164 INVX RTS
CC1D: 165 *
CC1D: 166 * PICK lifts a character from the screen in either
CC1D: 167 * 40 or 80 columns from the current cursor position.
CC1D: 168 * If the alternate character set is switched in,
CC1D: 169 * character codes $0-$1F are returned as $40-$5F (which
CC1D: 170 * is what must have been originally printed to the location).
CC1D: 171 *
CC1D:5A 172 PICKY PHY ;save Y
CC1E:20 9D CC 173 JSR GETCUR ;get newest cursor into Y
CC21:AD 1F C0 174 LDA RD80VID ;80 columns?
CC24:10 17 CC3D 175 BPL PICK1 ;=>no
CC26:8D 01 C0 176 STA SET80COL ;force 80STORE if 80 columns
CC29:98 177 TYA
CC2A:45 20 178 EOR WNDLFT ;C=1 if char in main RAM
CC2C:6A 179 ROR A ;get low bit into carry
CC2D:B0 04 CC33 180 BCS PICK2 ;=>store in main memory
CC2F:AD 55 C0 181 LDA TXTPAGE2 ;else switch in page 2
CC32:C8 182 INY ;for odd left, aux bytes
CC33:98 183 PICK2 TYA ;divide pos'n by 2
CC34:4A 184 LSR A
CC35:A8 185 TAY ;and use as offset into line
CC36:B1 28 186 LDA (BASL),Y ;pick character
CC38:8D 54 C0 187 STA TXTPAGE1 ;80 columns, switch in
CC3B:80 02 CC3F 188 BRA PICK3 ;skip 40 column pick
CC3D:B1 28 189 PICK1 LDA (BASL),Y ;pick 40 column char
CC3F:2C 1E C0 190 PICK3 BIT ALTCHARSET ;only allow if alt set
CC42:10 06 CC4A 191 BPL PICK4
CC44:C9 20 192 CMP #$20
CC46:B0 02 CC4A 193 BCS PICK4
CC48:09 40 194 ORA #$40
CC4A:7A 195 PICK4 PLY ;restore real Y
CC4B:60 196 RTS
CC4C: 197 *
CC4C: 198 * SHOWCUR displays either a checkerboard cursor, a solid
CC4C: 199 * rectangle, or the current cursor character, depending
CC4C: 200 * on the value of the CURSOR location. 0=inverse cursor,
CC4C: 201 * $FF=checkerboard cursor, anything else is displayed
CC4C: 202 * after being anded with inverse mask.

```

```

CC4C:      203 *
CC4C:AC FB 07 204 SHOWCUR LDY CURSOR ;what's my type?
CC4F:D0 02 CC53 205 BNE NOTINV ;=>not inverse
CC51:80 BF CC12 206 BRA INVERT ;else invert the char (exit)
CC53:      207 *
CC53:      208 * Exit with char in accumulator
CC53:      209 *
CC53:20 1D CC 210 NOTINV JSR PICKY ;get char on screen
CC56:48      211 PHA ;preserve it
CC57:8D 7B 07 212 STA NXTCUR ;save for update
CC5A:98      213 TYA ;test for checkerboard
CC5B:C8      214 INY
CC5C:F0 0D CC6B 215 BEQ NOTINV2 ;=>checkerboard, display it
CC5E:7A      216 PLY ;test char
CC5F:5A      217 PHY
CC60:30 09 CC6B 218 BMI NOTINV2 ;don't need inverse
CC62:AD 1E C0 219 LDA ALTCHARSET ;mask = $7F if alternate
CC65:09 7F 220 ORA #$7F ; character set,
CC67:4A      221 LSR A ;$3F if normal char set
CC68:2D FB 07 222 NOTINV1 AND CURSOR ;form char to display
CC6B:20 B3 C3 223 NOTINV2 JSR STORY ;and display it
CC6E:68      224 PLA ;restore real char
CC6F:60      225 RTS
CC70:      226 *
CC70:      227 * The UPDATE routine increments the random seed.
CC70:      228 * If a certain value is reached and we are in Apple II
CC70:      229 * mode, the blinking check cursor is updated. If a
CC70:      230 * key has been pressed, the old char is replaced on the
CC70:      231 * screen, and we return with BMI.
CC70:      232 *
CC70:      233 * NOTE: this routine used by COMM firmware!!
CC70:      234 *
CC70:48      235 UPDATE PHA ;save char
CC71:E6 4E      236 INC RNDL ;update seed
CC73:D0 1C CC91 237 BNE UD2 ;check for key
CC75:A5 4F      238 LDA RNDH
CC77:E6 4F      239 INC RNDH
CC79:45 4F      240 EOR RNDH
CC7B:29 10      241 AND #$10 ;need to update cursor?
CC7D:F0 12 CC91 242 BEQ UD2 ;=>no, check for key
CC7F:AD FB 07 243 LDA CURSOR ;what cursor are we using?
CC82:F0 0D CC91 244 BEQ UD2 ;=>//e cursor, leave alone
CC84:20 1D CC 245 JSR PICKY ;get the character into A
CC87:AC 7B 07 246 LDY NXTCUR ;get next character
CC8A:8D 7B 07 247 STA NXTCUR ;save next next character
CC8D:98      248 TYA
CC8E:20 B3 C3 249 JSR STORY ;and print it
CC91:68      250 UD2 PLA ;get real char
CC92:20 AD C9 251 JSR XBTKBD ;was a key pressed?
CC95:10 28 CCBF 252 BPL GETCURX ;=>no key pressed
CC97:20 B3 C3 253 CLRKBD JSR STORY ;restore old key
CC9A:4C 8F C9 254 JMP XRDKBD ;look for keystroke and exit
CC9D:      255 *
CC9D:      256 * ON CURSORS. Whenever the horizontal cursor position is
CC9D:      257 * needed, a call to GETCUR is done. This is the equivalent
CC9D:      258 * of a LDY CH. This returns the current cursor for II and
CC9D:      259 * //e mode, which may have been poked as either CH or CURCH.
CC9D:      260 *

```

```

CC9D:      261 * It also forces CH and OLDCH to 0 if 80 column mode active.
CC9D:      262 * This prevents LDY CH, STA (BASL),Y from trashing non screen
CC9D:      263 * memory. It works just like the //e.
CC9D:      264 *
CC9D:      265 * All routines that update the cursor's horizontal position
CC9D:      266 * are here. This ensures that the newest value of the cursor
CC9D:      267 * is always used, and that 80 column CH is always 0.
CC9D:      268 *
CC9D:      269 * GETCUR only affects the Y register
CC9D:      270 *
CC9D:A4 24 271 GETCUR    LDY    CH            ;if CH=OLDCH, then
CC9F:CC 7B 04 272          CPY    OLDCH          ;OURCH is valid
CCA2:D0 03 CCA7 273          BNE    GETCUR1        ;=>else CH must have been changed
CCA4:AC 7B 05 274          LDY    OURCH          ;use OURCH
CCA7:C4 21 275 GETCUR1    CPY    WNDWIDTH      ;is the value too big
CCA9:90 02 CCAD 276          BCC    GETCUR2        ;=>no, fits just fine
CCAB:A0 00 277          LDY    #0            ;else force CH to 0
CCAD:      278 *
CCAD:      279 * GETCUR2 is commonly used to set the current cursor
CCAD:      280 * position when Y can be used.
CCAD:      281 *
CCAD:8C 7B 05 282 GETCUR2    STY    OURCH          ;update real cursor
CCB0:2C 1F C0 283          BIT    RDB0VID        ;80 columns?
CCB3:10 02 CCB7 284          BPL    GETCUR3        ;=>no, set all cursors
CCB5:A0 00 285          LDY    #0            ;yes, peg CH to 0
CCB7:84 24 286 GETCUR3    STY    CH
CCB9:8C 7B 04 287          STY    OLDCH
CCBC:AC 7B 05 288          LDY    OURCH          ;get cursor
CCBF:60 289 GETCURX    RTS            ;and fly...
CCCO:      28 INCLUDE ESCAPE

```

```

CCCC:      2 * START AN ESCAPE SEQUENCE:
CCCC:      3 * WE HANDLE THE FOLLOWING ONES:
CCCC:      4 *   @ - HOME & CLEAR
CCCC:      5 *   A - Cursor right
CCCC:      6 *   B - Cursor left
CCCC:      7 *   C - Cursor down
CCCC:      8 *   D - Cursor up
CCCC:      9 *   E - CLR TO EOL
CCCC:     10 *   F - CLR TO EOS
CCCC:     11 *   I, Up Arrow - CURSOR UP (stay escape)
CCCC:     12 *   J, Lft Arrow - CURSOR LEFT (stay escape)
CCCC:     13 *   K, Rt Arrow - CURSOR RIGHT (stay escape)
CCCC:     14 *   M, Dn Arrow - CURSOR DOWN (stay escape)
CCCC:     15 *   4 - GOTO 40 COLUMN MODE
CCCC:     16 *   8 - GOTO 80 COLUMN MODE
CCCC:     17 * CTL-D- Disable the printing of control chars
CCCC:     18 * CTL-E- Enable the printing of control chars
CCCC:     19 * CTL-Q- QUIT (PR#0/IN#0)
CCCC:     20 *
CCCC:B9 0C CD 21 ESC3      LDA  ESCCHAR,Y      ;GET CHAR TO "PRINT"
CCCC:5A      22          PHY                      ;save index
CCCC:20 58 CD 23          JSR  CTLCHAR          ;execute character
CCCC:7A      24          PLY                      ;restore index
CCCC:C0 08    25          CPY  #YHI             ;If Y<YHI, stay escape
CCCC:A0 21 CCED 26          BCS  ESCRDKEY        ;=>exit escape mode
CCCC:      27 *
CCCC:      28 * This is the entry point called by RDKEY iff escapes
CCCC:      29 * are enabled and an escape is encountered. The next
CCCC:      30 * keypress is read and processed. If it is a key that
CCCC:      31 * terminates escape mode, a new key is read by ESCRDKEY.
CCCC:      32 * If escape mode should not be terminated, NEWESC is
CCCC:      33 * called again.
CCCC:      34 *
CCCC:20 1D CC 35 NEWESC    JSR  PICKY             ;get current character
CCCC:48      36          PHA                      ;and save it
CCCC:29 80    37          AND  #$80             ;save invert bit
CCCC:49 AB    38          EOR  #$AB             ;make it inverted "+"
CCCC:20 B3 C3 39          JSR  STORY            ;and pop it on the screen
CCCC:20 AD C9 40 ESC0      JSR  XBTKBD          ;check for keystroke
CCCC:A0 18 CCD7 41          BPL  ESC0
CCCC:68      42          PLA                      ;get old char
CCCC:20 97 CC 43          JSR  CLRKBD          ;restore char, get key
CCCC:20 9B C3 44          JSR  UPSHIFT         ;upshift esc char
CCCC:A0 13    45 ESC1      LDY  #ESCTAB        ;COUNT/INDEX
CCCC:D9 F8 CC 46 ESC2      CMP  ESCTAB,Y       ;IS IT A VALID ESCAPE?
CCCC:F0 D6 CCC0 47          BEQ  ESC3           =>yes
CCCC:A8      48          DEY
CCCC:10 F8 CCE5 49          BPL  ESC2           ;TRY 'EM ALL...
CCCC:      50 *
CCCC:      51 * End of escape sequence, read next character.
CCCC:      52 * This is initially called by RDCHAR which is usually called
CCCC:      53 * by GETLN to read characters with escapes enabled.
CCCC:      54 *
CCCC:A9 08    55 ESCRDKEY  LDA  #M.CTL          ;enable escape sequences
CCCC:F1C FB D4 56          TRB  VMODE
CCCC:20 0C FD 57          JSR  RDKEY            ;read char with escapes
CCCC:4C 44 FD 58          JMP  NOESCAPE        ;got the key, disable escapes
CCCC:      59 *

```

```

CCF8:      60 * When in escape mode, the characters in ESGTAB (high)
CCF8:      61 * bits set), are mapped into the characters in ESCCHAR.
CCF8:      62 * These characters are then executed by a call to CTLCHAR.
CCF8:      63 *
CCF8:      64 * CTLCHAR looks up a character in the table starting at
CCF8:      65 * CTLTAB. It uses the current index as an index into the
CCF8:      66 * table of routine addresses, CTLADR. If the character is
CCF8:      67 * not in the table, a call to VIDOUT1 is done in case the
CCF8:      68 * character is BS, LF, CR, or BEL.
CCF8:      69 *
CCF8:      70 * NOTE: CILON and CTLOFF are not accessible except through
CCF8:      71 * and escape sequence
CCF8:      72 *
CCF8:      73          MSB   ON          ;high bit on
CCF8:      CCF8 74 ESGTAB   EQU   *
CCF8:CA     75          ASC   'J'   ;left (stay esc)
CCF8:8B     76          DFB   $88   ;left arrow (stay esc)
CCF8:CD     77          ASC   'M'   ;down (stay esc)
CCF8:8B     78          DFB   $8B   ;up arrow (stay esc)
CCF8:95     79          DFB   $95   ;right arrow (stay esc)
CCF8:8A     80          DFB   $8A   ;down arrow (stay esc)
CCF8:C9     81          ASC   'I'   ;up (stay esc)
CCF8:CB     82          ASC   'K'   ;right (stay esc)
CD00:      0008 83 YHI     EQU   *-ESGTAB
CD00:C2     84          ASC   'B'   ;left
CD01:C3     85          ASC   'C'   ;down
CD02:C4     86          ASC   'D'   ;up
CD03:C1     87          ASC   'A'   ;right
CD04:C0     88          ASC   '@'   ;formfeed
CD05:C5     89          ASC   'E'   ;clear EOL
CD06:C6     90          ASC   'F'   ;clear EOS
CD07:B4     91          ASC   '4'   ;40 column mode
CD08:B8     92          ASC   '8'   ;80 column mode
CD09:91     93          DFB   $91   ;CTL-Q = QUIT
CD0A:84     94          DFB   $84   ;CTL-D ;ctl char disable
CD0B:85     95          DFB   $85   ;CTL-E ;ctl char enable
CD0C:      96 *
CD0C:      0013 97 ESCNUM   EQU   *-ESGTAB-1
CD0C:      98 *
CD0C:      CDOC 99 ESCCHAR  EQU   *          ;list of escape chars
CD0C:88     100          DFB   $88   ;J: BS (stay esc)
CD0D:88     101          DFB   $88   ;<-:BS (stay esc)
CD0E:8A     102          DFB   $8A   ;M: LF (stay esc)
CD0F:9F     103          DFB   $9F   ;UP:US (stay esc)
CD10:9C     104          DFB   $9C   ;->:FS (stay esc)
CD11:8A     105          DFB   $8A   ;DN: LF (stay esc)
CD12:9F     106          DFB   $9F   ;I: UP (stay esc)
CD13:9C     107          DFB   $9C   ;K: RT (stay esc)
CD14:88     108          DFB   $88   ;ESC-8 = BS
CD15:      CD15 109 CTLTAB  EQU   *          ;list of control characters
CD15:8A     110          DFB   $8A   ;ESC-C = DN
CD16:9F     111          DFB   $9F   ;ESC-D = UP
CD17:9C     112          DFB   $9C   ;ESC-A = RT
CD18:8C     113          DFB   $8C   ;@: Formfeed
CD19:9D     114          DFB   $9D   ;E: CLREOL
CD1A:8B     115          DFB   $8B   ;F: CLREOP
CD1B:91     116          DFB   $91   ;SET40
CD1C:92     117          DFB   $92   ;SET80

```



CD1D:95	118	DFB	\$95	;QUIT
CD1E:04	119	DFB	\$04	;Disable controls (escape only)
CD1F:05	120	DFB	\$05	;Enable controls (escape only)
CD20:	121	* escape chars end here		
CD20:85	122	DFB	\$85	;X.CUR.ON
CD21:86	123	DFB	\$86	;X.CUR.OFF
CD22:8E	124	DFB	\$8E	;Normal
CD23:8F	125	DFB	\$8F	;Inverse
CD24:96	126	DFB	\$96	;Scroll down
CD25:97	127	DFB	\$97	;Scroll up
CD26:98	128	DFB	\$98	;mouse chars off
CD27:99	129	DFB	\$99	;home cursor
CD28:9A	130	DFB	\$9A	;clear line
CD29:9B	131	DFB	\$9B	;mouse chars on
CD2A:	132	*		
CD2A:	0014 133	CTLNUM	EQU	*-CTLTAB-1
CD2A:	134	*		
CD2A:	CD2A 135	CTLADR	EQU	*
CD2A:66 FC	136	DW	LF	;move cursor down
CD2C:1A FC	137	DW	UP	;move cursor up
CD2E:A0 FB	138	DW	NEWADV	;forward a space
CD30:58 FC	139	DW	HOME	;home cursor, clear screen
CD32:9C FC	140	DW	CLREOL	;clear to end of line
CD34:42 FC	141	DW	CLREOP	;clear to end of page
CD36:C0 CD	142	DW	SET40	;set 40 column mode
CD38:BE CD	143	DW	SET80	;set 80 column mode
CD3A:45 CE	144	DW	QUIT	;Quit video firmware
CD3C:91 CD	145	DW	CTLOFF	;disable //e control chars
CD3E:95 CD	146	DW	CTLON	;enable //e control chars
CD40:89 CD	147	DW	X.CUR.ON	;turn on cursor (pascal)
CD42:8D CD	148	DW	X.CUR.OFF	;turn off cursor (pascal)
CD44:B0 CD	149	DW	X.S0	;normal video
CD46:B7 CD	150	DW	X.S1	;inverse video
CD48:30 CB	151	DW	SCROLLDN	;scroll down a line
CD4A:35 CB	152	DW	SCROLLUP	;scroll up a line
CD4C:9F CD	153	DW	MOUSOFF	;disable mouse characters
CD4E:A5 CD	154	DW	HOMECUR	;move cursor home
CD50:A0 FC	155	DW	CLRLIN	;clear current line
CD52:99 CD	156	DW	MOUSON	;enable mouse characters
CD54:	157	*		
CD54:	158	MSB	ON	
CD54:	159	*		
CD54:	160	* CTLCHAR executes the control character in the		
CD54:	161	* accumulator. If it is called by Pascal, the character		
CD54:	162	* is always executed. If it is called by the video		
CD54:	163	* firmware, the character is executed if M.CTL is set		
CD54:	164	* and M.CTL2 is clear.		
CD54:	165	*		
CD54:	166	* Note: This routine is only called if the video firmware		
CD54:	167	* is active. The Monitor ROM calls VIDOUT! if the video		
CD54:	168	* firmware is inactive.		
CD54:	169	*		
CD54:2C C1 C8	170	CTLCHAR0	BIT	SEV1 ;set V (use M.CTL)
CD57:50	171	DFB	\$50	;BVC opcode (never taken)
CD58:	172	*		
CD58:B8	173	CTLCHAR	CLV	;Always do control character
CD59:DA	174	PHX		;save X
CD5A:8D F8 04	175	STA	TEMP1	;temp save of A



CD5D:20 04 FC	176	JSR	VIDOUT1	;try to execute CR, LF, BS, or BEL
CD60:CD F8 04	177	CMP	TEMP1	;if acc has changed
CD63:DD 0A CD6F	178	BNE	CTLDONE	;then function done
CD65:A2 14	179	LDX	#CTLNUM	;number of CTL chars
CD67:DD 15 CD	180	CMP	CTLTAB,X	;is it in table
CD6A:F0 05 CD71	181	BEQ	CTLGO	;=>yes, should we execute?
CD6C:CA	182	DEX		;else check next
CD6D:10 F8 CD67	183	BFL	FNDCTL	;=>try next one
CD6F:FA	184	PLX		;restore X
CD70:60	185	RTS		;and return
CD71:	186	*		
CD71:48	187	CTLGO	PRA	;save A
CD72:50 0C CD80	188	BVC	CTLGO1	;V clear, always do (pascal,escape)
CD74:AD FB 04	189	LDA	VMODE	;controls are enabled iff
CD77:29 28	190	AND	#M.CTL+M.CTL2	; M.CTL = 1 and
CD79:49 08	191	EOR	#M.CTL	; M.CTL2 = 0
CD7B:F0 03 CD80	192	BEQ	CTLGO1	;=>they're enabled!!
CD7D:68	193	CGO	PLA	;restore A
CD7E:FA	194	PLX		;restore X
CD7F:60	195	RTS		;and return
CD80:	196	*		
CD80:8A	197	CTLGO1	TXA	;double X as index
CD81:0A	198	ASL	A	;into address table
CD82:AA	199	TAX		
CD83:68	200	PLA		;restore A
CD84:20 A4 FC	201	JSR	CTLDO	;execute the char
CD87:FA	202	PLX		;restore X
CD88:60	203	RTS		;and return
CD89:	204	*		
CD89:	205	*	X.CUR.ON = Allow Pascal cursor display	
CD89:	206	*	X.CUR.OFF = Disable Pascal cursor display	
CD89:	207	*	Cursor is not displayed during call, so it will	
CD89:	208	*	be right when "redisplayed".	
CD89:	209	*	Note: Though these commands are executed from BASIC,	
CD89:	210	*	they have no effect on firmware operation.	
CD89:	211	*		
CD89:A9 10	212	X.CUR.ON LDA	#M.CURSOR	;clear cursor bit
CD8B:80 0E CD9B	213	BRA	CLRIT	
CD8D:	214	*		
CD8D:A9 10	215	X.CUR.OFF LDA	#M.CURSOR	;set cursor bit
CD8F:80 10 CDA1	216	BRA	SETIT	
CD91:	217	*		
CD91:	218	*	The control characters other than CR,LF,BEL,BS	
CD91:	219	*	are normally enabled when video firmware is active.	
CD91:	220	*	They can be disabled and enabled using the ESC-D	
CD91:	221	*	and ESC-E escape sequences.	
CD91:	222	*		
CD91:A9 20	223	CTLOFF LDA	#M.CTL2	;disable control characters
CD93:80 0C CDA1	224	BRA	SETIT	;by setting M.CTL2
CD95:	225	*		
CD95:A9 20	226	CTLON LDA	#M.CTL2	;enable control characters
CD97:80 02 CD9B	227	BRA	CLRIT	;by clearing M.CTL2
CD99:	228	*		
CD99:	229	*	Enable mouse text by clearing M.MOUSE	
CD99:	230	*		
CD99:A9 01	231	MOUSON LDA	#M.MOUSE	
CD9B:1C FB 04	232	CLRIT	TRB VMODE	
CD9E:60	233	RTS		

```

CD9F:          234 *
CD9F:          235 * Disable mouse text by setting M.MOUSE
CD9F:          236 *
CD9F:A9 01     237 MOUSOFF   LDA    #M.MOUSE
CDA1:0C FB 04   238 SETIT    TSB    VMODE
CDA4:60         239          RTS
CDA5:          240 *
CDA5:          241 * EXECUTE HOME:
CDA5:          242 *
CDA5:20 E9 FE   243 HOMECUR   JSR    CLRCH      ;move cursors to far left
CDA8:A8         244          TAY          ;(probably not needed)
CDA9:A5 22      245          LDA    WNDTOP    ;and to top of window
CDAB:85 25      246          STA    CV
CDAD:4C 88 FC   247          JMP    NEWVTABZ    ;then set base address, OURCV
CDB0:          248 *
CDB0:          249 * EXECUTE "NORMAL VIDEO"
CDB0:          250 *
CDB0:20 B4 FE   251 X.SO      JSR    SETNORM    ;set INVFLG to $FF
CDB3:A9 04      252          LDA    #M.VMODE    ;then clear inverse mode bit
CDB5:80 E4      253          BRA    CLRIT
CDB7:          254 *
CDB7:          255 * EXECUTE "INVERSE VIDEO"
CDB7:          256 *
CDB7:20 80 FE   257 X.SI      JSR    SETINV     ;set INVFLG to $3F
CDEA:A9 04      258          LDA    #M.VMODE    ;then set inverse mode bit
CDEB:80 E3      259          BRA    SETIT
CDBE:          260 *
CDBE:          261 * EXECUTE '40COL MODE' or '80COL MODE':
CDBE:          262 *
CDBE:38         263 SET80      SEC          ;flag an 80 column window
CDBF:90         264          DFB    $90      ;BCC opcode (never taken)
CDC0:18         265 SET40      CLC          ;flag a 40 column window
CDC1:2C FB 04   266          BIT    VMODE      ;but...is it pascal?
CDC4:10 54      267          BPL    SETX     ;=>yes, don't execute
CDC6:08         268          PHP          ;save window size
CDC7:20 1B CE   269          JSR    HOOKITUP    ;COPYROM if needed, set I/O hooks
CDCA:28         270          PLP          ;and get 40/80
CDCB:80 08      271          BRA    WIND      ;=>set window
CDCB:          272 *
CDCD:          273 * CHK80 is called by PR#0 to convert to 40 if it was
CDCD:          274 * 80. Otherwise the window is left ajar.
CDCD:          275 *
CDCD:2C 1F C0   276 CHK80      BIT    RD8OVID    ;don't set 40 if
CDD0:10 48      277          BPL    SETX     ;already 40
CDD2:          278 *
CDD2:18         279 WIN40      CLC          ;flag 40 column window
CDD3:80         280          DFB    $B0      ;BCS opcode (never taken)
CDD4:38         281 WIN80      SEC          ;flag 80 column window
CDD5:64 22      282 WIN0      STZ    WNDTOP    ;set window top now
CDD7:2C 1A C0   283          BIT    RDTEXT     ;for text or mixed
CDDA:30 04      284          BMI    WIN1     ;=>text
CDDC:A9 14      285          LDA    #20
CDEB:85 22      286          STA    WNDTOP    ;used by 80<->40 conversion
CDE0:2C 1F C0   287 WIN1      BIT    RD8OVID    ;80 columns now?
CDE3:08         288          PHP          ;save 80 or 40
CDE4:80 07      289          BCS    WIN2     ;=>80: convert if 40
CDE6:10 0A      290          BPL    WIN3     ;=>40: no convert
CDE8:20 53 CE   291          JSR    SCRNB4     ;80: convert to 40

```

```

CDEB:80 05 CDF2 292      BRA WIN3      ;done converting
CDED:30 03 CDF2 293 WIN2 BMI WIN3      ;=>80: no convert
CDEF:20 80 CE 294      JSR SCRN48      ;40: convert to 80
CDF2:20 9D CC 295 WIN3 JSR GETCUR      ;determine absolute CH
CDF5:98 296      TYA      ;in case the window setting
CDF6:18 297      CLC      ;was different
CDF7:65 20 298      ADC WNDLFT
CDF9:28 299      PLP      ;pin to right edge if
CDFA:B0 06 CE02 300 ECS WIN4      ;80 to 40 leaves cursor
CDFC:C9 28 301      CMP #40      ;off the screen
CDFE:90 02 CE02 302      BCC WIN4
CE00:A9 27 303      LDA #39
CE02:20 EC FE 304 WIN4 JSR SETCUR      ;set new cursor
CE05:A5 25 305      LDA CV      ;set new base address
CE07:20 C1 FB 306      JSR BASCALC      ;for left = 0 (always)
CE0A: 307 *
CE0A:64 20 308 WNDREST STZ WNDLFT      ;Called by INIT and Pascal
CE0C:A9 18 309      LDA #18      ;and bottom
CE0E:85 23 310      STA WNDBTM
CE10:A9 28 311      LDA #28      ;set left,width,bottom
CE12:2C 1F C0 312      BIT RD80VID      ;set width to 80 if 80 columns
CE15:10 01 CE18 313      BPL WIN5
CE17:0A 314      ASL A
CE18:85 21 315 WIN5 STA WNDWDTH      ;set width
CE1A:60 316 SETX RTS      ;exit used by SET40/80
CE1B: 317 *
CE1B: 318 * Turn on video firmware;
CE1B: 319 *
CE1B: 320 * This routine is used by BASIC init, ESC-4, ESC-8
CE1B: 321 * It copies the Monitor ROM to the language card
CE1B: 322 * if necessary; it sets the input and output hooks to
CE1B: 323 * $C30x; it sets all switches for video firmware operation
CE1B: 324 *
CE1B:2C 7B 06 325 HOOKITUP BIT VFACTV      ;don't touch hooks
CE1E:10 11 CE31 326      BPL VIDMODE      ;if video firmware already active
CE20:20 38 C3 327 HOOKUP JSR COPYROM      ;Copy ROM to LC7
CE23:A9 05 328 SETHOOKS LDA #>C3KEYIN      ;set up $C300 hooks
CE25:85 38 329      STA KSWL
CE27:A9 07 330      LDA #>C3COUT1
CE29:85 36 331      STA CSWL
CE2B:A9 C3 332      LDA #<C3COUT1
CE2D:85 39 333      STA KSWH
CE2F:85 37 334      STA CSWH
CE31: 335 *
CE31: 336 * Now set the video firmware active
CE31: 337 *
CE31:9C FB 07 338 VIDMODE STZ CURSOR      ;set a solid inverse cursor
CE34:A9 08 339      LDA #M.CTL      ;preserve M.CTL bit
CE36:2D FB 04 340      AND VMODE
CE39:09 81 341      ORA #M.PASCAL+M.MOUSE ;no pascal,mouse
CE3B: 342 *
CE3B: 343 * Pascal calls here to set its mode
CE3B: 344 *
CE3B:8D FB 04 345 PVMODE STA VMODE      ;set mode bits
CE3E:9C 7B 06 346      STZ VFACTV      ;say video firmware active
CE41:8D 0F C0 347      STA SETALTCHAR ;and set alternate char set
CE44:60 348 OX RTS
CE45: 349 *

```

```

CE45:          350 * QUIT converts the screen from 80 to 40 if necessary,
CE45:          351 * sets a 40 column window, and restores the normal I/O
CE45:          352 * hooks (COUTI and KEYIN).
CE45:          353 *
CE45:2C FB 04   354 QUIT      BIT    VMODE      ;no quitting from pascal
CE48:10 FA CE44 355          BPL    QX
CE4A:20 D2 CD   356          JSR    WIN40      ;first, do an escape 4
CE4D:20 89 FE   357 ZZQUIT   JSR    SETKBD     ;do a IN#0 (used by COMM)
CE50:4C 93 FE   358          JMP    SETVID     ;and a PR#0

```

```

CE53:          360 *
CE53:          361 * SCRNB4 and SCRNB48 convert screens between 40 & 80 cols.
CE53:          362 * WNDTOP must be set up to indicate the last line to
CE53:          363 * be done. All registers are trashed.
CE53:          364 *
CE53:A2 17      365 SCRNB4      LDX    #23          ;start at bottom of screen
CE55:8D 01 C0    366          STA    SET80COL      ;allow page 2 access
CE58:8A          367 SCR1       TXA          ;calc base for line
CE59:2D C1 FB    368          JSR    BASCALC
CE5C:A0 27      369          LDY    #39          ;start at right of screen
CE5E:5A          370 SCR2       PHY          ;save 40 index
CE5F:98          371          TYA          ;div by 2 for 80 column index
CE60:4A          372          LSR    A
CE61:B0 03 CE66  373          BCS    SCR3
CE63:2C 55 C0    374          BIT    TXTPAGE2      ;even column, do page 2
CE66:A8          375 SCR3       TAY          ;get 80 index
CE67:B1 28      376          LDA    (BASL),Y      ;get 80 char
CE69:2C 54 C0    377          BIT    TXTPAGE1      ;restore pagel
CE6C:7A          378          PLY          ;get 40 index
CE6D:91 28      379          STA    (BASL),Y
CE6F:88          380          DEY
CE70:10 EC CE5E  381          BPL    SCR2          ;do next 40 byte
CE72:CA          382          DEX          ;do next line
CE73:3D 04 CE79  383          BMI    SCR4          ;=>done with setup
CE75:E4 22      384          CPX    WNDTOP        ;at top yet?
CE77:B0 DF CE58  385          BCS    SCR1
CE79:8D 00 C0    386 SCR4      STA    CLR80COL      ;clear 80STORE for 40 columns
CE7C:8D 0C C0    387          STA    CLR80VID      ;clear 80VID for 40 columns
CE7F:60          388          RTS
CE80:          389 *
CE80:A2 17      390 SCRNB48    LDX    #23          ;start at bottom of screen
CE82:8A          391 SCR5       TXA          ;set base for current line
CE83:2D C1 FB    392          JSR    BASCALC
CE86:A0 00      393          LDY    #0          ;start at left of screen
CE88:8D 01 C0    394          STA    SET80COL      ;enable page2 store
CE8B:B1 28      395 SCR6       LDA    (BASL),Y      ;get 40 column char
CE8D:5A          396 SCR8       PHY          ;save 40 column index
CE8E:48          397          PHA          ;save char
CE8F:98          398          TYA          ;div 2 for 80 column index
CE90:4A          399          LSR    A
CE91:B0 03 CE96  400          BCS    SCR7          ;save on pagel
CE93:8D 55 C0    401          STA    TXTPAGE2
CE96:A8          402 SCR7       TAY          ;get 80 column index
CE97:68          403          PLA          ;now save character
CE98:91 28      404          STA    (BASL),Y
CE9A:8D 54 C0    405          STA    TXTPAGE1      ;flip pagel
CE9D:7A          406          PLY          ;restore 40 column index
CE9E:CB          407          INY          ;move to the right
CE9F:C0 28      408          CPY    #40          ;at right yet?
CEA1:90 E8 CE8B  409          BCC    SCR6          ;=>no, do next column
CEA3:2D CF CB    410          JSR    CLRHALF        ;clear half of screen
CEA6:CA          411          DEX          ;else do next line of screen
CEA7:3D 04 CEAD  412          BMI    SCR9          ;=>done with top line
CEA9:E4 22      413          CPX    WNDTOP        ;at top yet?
CEAB:B0 D5 CE82  414          BCS    SCR5
CEAD:8D 0D C0    415 SCR9      STA    SET80VID      ;convert to 80 columns
CEB0:60          416          RTS
CEB1:          29          INCLUDE PASCAL      ;Pascal support stuff

```



CEB1:AA		3	PSTATUS	TAX		;is request code = 0?
CEB2:F0 08	CEBC	4		BEQ	PIORDY	;=>yes, ready for output
CEB4:CA		5		DEX		;check for any input
CEB5:D0 07	CEBE	6		BNE	PSTERR	;=>bad request, return error
CEB7:20 AD C9		7		JSR	XBITKBD	;test keyboard
CEBA:10 04	CEC0	8		BPL	PNOTRDY	;=>no keystroked
CEBC:38		9	PIORDY	SEC		;good return
CEBD:60		10		RTS		
CEBE:A2 03		11	PSTERR	LDX	#3	;else flag error
CEC0:18		12	PNOTRDY	CLC		
CEC1:60		13		RTS		
CEC2:		14	*			
CEC2:		15	* PASCAL OUTPUT:			
CEC2:		16	*			
CEC2:	CEC2	17	PWRITE	EQU	*	
CEC2:09 80		18		ORA	#\$80	;turn on high bit
CEC4:AA		19		TAX		;save character
CEC5:20 54 CF		20		JSR	PSETUP2	;SETUP ZP STUFF, don't set ROM
CEC8:A9 08		21		LDA	#M.GOXY	;ARE WE DOING GOTOXY?
CECA:2C FB 04		22		BIT	VMODE	
CECD:D0 2B	CEFA	23		BNE	GETX	;=>Doing X or Y?
CECF:8A		24		TXA		;now check for control char
CEEO:89 60		25		BIT	#\$60	;is it control?
CEED2:F0 45	CF19	26		BEQ	PCTL	;=>yes, do control
CEED4:AC 7B 05		27		LDY	OURCH	;get horizontal position
CEED7:24 32		28		BIT	INVFLG	;check for inverse
CEED9:30 02	CEED	29		BMI	PWR1	;normal, go store it
CEEDB:29 7F		30		AND	#\$7F	
CEEDD:20 C1 C3		31	PWR1	JSR	STORE	;now store it (erasing cursor)
CEED0:C8		32		INY		;INC CH



CEE1:8C 7B 05	33	STY	OURCH	
CEE4:C4 21	34	CPY	WNDWDTH	
CEE6:90 0C CEF4	35	BCC	PWRET	
CEE8:20 60 C3	36	JSR	SETROM	
CEE8:20 E9 FE	37	JSR	CLRCM	;set cursor position to 0
CEE2:20 66 FC	38	JSR	LF	
CEF1:20 54 C3	39	PWRITERET JSR	RESETLC	
CEF4:20 0B CC	40	PWRET JSR	PASINVERT	;display new cursor
CEF7:A2 00	41	PRET LDX	#S0	;return with no error
CEF9:60	42	RTS		
CEFA:	43	*		
CEFA:	44	* HANDLE GOTOXY STUFF;		
CEFA:	45	*		
CEFA: CEF4	46	GETX EQU	*	
CEFA:20 0B CC	47	JSR	PASINVERT	;turn off cursor
CEFD:8A	48	TXA		;get character
CEFE:38	49	SEC		
CEFF:E9 A0	50	SBC	#160	;MAKE BINARY
CF01:2C FB 06	51	BIT	XCOORD	;doing X?
CF04:30 2A CF30	52	BMI	PSETX	;=>yes, set it
CF06:	53	*		
CF06:	54	* Set Y and do the GOTOXY		
CF06:	55	*		
CF06: CF06	56	GETY EQU	*	
CF06:8D FB 05	57	STA	OURCV	
CF09:20 71 CF	58	JSR	PASCALC	;calc base addr
CF0C:AC FB 06	59	LDY	XCOORD	
CF0F:20 AD CC	60	JSR	GETCUR2	;set proper cursors

```

CF12:A9 08      61      LDA      #M.GOXY      ;turn off gotoxy
CF14:1C FB 04      62      TRB      VMODE
CF17:80 DB      CEF4    63      BRA      PWRET      ;=>DONE (ALWAYS TAKEN)
CF19:          64 *
CF19:20 0B CC      65 PCTL      JSR      PASINVERT    ;turn off cursor
CF1C:8A          66      TXA          ;get char
CF1D:C9 9E          67      CMP      #$9E      ;is it gotoXY?
CF1F:F0 08      CF29    68      BEQ      STARTXY    ;=>yes, start it up
CF21:20 60 C3      69      JSR      SETROM      ;must switch in ROM for controls
CF24:20 58 CD      70      JSR      CTLCHAR      ;EXECUTE IT IF POSSIBLE
CF27:80 C8      CEF1    71      BRA      PWRITERET    ;=>display new cursor, exit
CF29:          72 *
CF29:          73 * START THE GOTOXY SEQUENCE:
CF29:          74 *
CF29:          CF29    75 STARTXY    EQU      *
CF29:A9 08      76      LDA      #M.GOXY
CF2B:0C FB 04      77      TSB      VMODE      ;turn on gotoxy
CF2E:A9 FF      78      LDA      #$FF      ;set XCOORD to -1
CF30:8D FB 06      79 PSETX      STA      XCOORD      ;set X
CF33:80 BF      CEF4    80      BRA      PWRET      ;=>display cursor and exit
CF35:          81 *
CF35:          82 * PASCAL INPUT:
CF35:          83 *
CF35:20 54 CF      84 PASREAD      JSR      PSETUP2      ;SETUP 2P STUFF
CF38:20 8F C9      85 GKEY          JSR      XRDKBD      ;key pressed?
CF3B:10 FB      CF38    86      BPL      GKEY      ;=>not yet
CF3D:29 7F      87      AND      #$7F      ;DROP Hi BIT
CF3F:80 B6      CEF7    88      BRA      PRET      ;good exit
CF41:          89 *
CF41:          90 * PASCAL INITIALIZATION:
CF41:          91 *
CF41:          CF41    92 PINIT      EQU      *
CF41:A9 01      93      LDA      #M.MOUSE      ;Set mode to pascal
CF43:20 3B CE      94      JSR      PVMODE      ;without mouse characters
CF46:20 51 CF      95      JSR      PSETUP      ;setup zero page for pascal
CF49:20 04 CD      96      JSR      WIN80      ;do 40->80 convert
CF4C:20 58 FC      97      JSR      HOME      ;home and clear screen
CF4F:80 A0      CEF1    98      BRA      PWRITERET    ;display cursor, set OURCH,OURCV...
CF51:          99 *
CF51:          CF51    100 PSETUP      EQU      *
CF51:20 60 C3      101      JSR      SETROM      ;save LC state, set ROM read
CF54:64 22      102 PSETUP2      STZ      WNDTOP      ;set top to 0
CF56:20 0A CE      103      JSR      WNDREST      ;init either 40 or 80 window
CF59:A9 FF      104      LDA      #$FF      ;assume normal text
CF5B:85 32      105      STA      INVFLG
CF5D:A9 04      106      LDA      #M.VMODE      ;is it
CF5F:2C FB 04      107      BIT      VMODE
CF62:F0 02      CF66    108      BEQ      PSI      ;=>yes
CF64:46 32      109      LSR      INVFLG      ;no, make flag inverse
CF66:AC 78 05      110 PSI          LDY      OURCH
CF69:20 AD CC      111      JSR      GETCUR2      ;set all cursors
CF6C:AD F8 05      112      LDA      OURCV
CF6F:B5 25      113      STA      CV
CF71:          114 *
CF71:          115 * Put BASCALC here so we don't have to switch
CF71:          116 * in the ROMs for each character output.
CF71:          117 *
CF71:0A          118 PASCALC      ASL      A

```

CF72:A8	119	TAY		;calc base addr in BASL,H
CF73:4A	120	LSR	A	;for given line no.
CF74:4A	121	LSR	A	
CF75:29 03	122	AND	#\$03	; 0<=line no.<=\$17
CF77:09 04	123	ORA	#\$4	; arg=000ABCDE, generate
CF79:85 29	124	STA	BASH	; BASH=000001CD
CF7B:98	125	TYA		;and
CF7C:5A	126	ROR	A	; BASL=EABAB000
CF7D:29 98	127	AND	#\$98	
CF7F:85 28	128	STA	BASL	
CF81:0A	129	ASL	A	
CF82:0A	130	ASL	A	
CF83:04 28	131	TSB	BASL	
CF85:60	132	RTS		
CF86:	30	INCLUDE AUXSTUFF		;Aux RAM routines

```

CF86:      4 *****
CF86:      5 * NAME      : MOVEAUX
CF86:      6 * FUNCTION: PERFORM CROSSBANK MEMORY MOVE
CF86:      7 * INPUT   : A1=SOURCE ADDRESS
CF86:      8 *       : A2=SOURCE END
CF86:      9 *       : A4=DESTINATION START
CF86:     10 *       : CARRY SET=MAIN-->CARD
CF86:     11 *       : CLR=CARD-->MAIN
CF86:     12 * OUTPUT  : NONE
CF86:     13 * VOLATILE: NOTHING
CF86:     14 * CALLS   : NOTHING
CF86:     15 *****
CF86:      16 MOVEAUX EQU *
CF86:48      17 PHA          ;SAVE AC
CF87:AD 13 CO 18 LDA RDRAMRD ;SAVE STATE OF
CF8A:48      19 PHA          ; MEMORY FLAGS
CF8B:AD 14 CO 20 LDA RDRAMWRT
CF8E:48      21 PHA
CF8F:      22 *
CF8F:      23 * SET FLAGS FOR CROSSBANK MOVE:
CF8F:      24 *
CF8F:90 08 CF99 25 BCC MOVEC2M ;=>CARD-->MAIN
CF91:8D 02 CO 26 STA RDMAINRAM ;SET FOR MAIN
CF94:8D 05 CO 27 STA WRCARDRAM ; TO CARD
CF97:B0 06 CF9F 28 BCS MOVESTRT ;=>(ALWAYS TAKEN)
CF99:      29 *
CF99:      30 MOVEC2M EQU *
CF99:8D 04 CO 31 STA WRMAINRAM ;SET FOR CARD
CF9C:8D 03 CO 32 STA RDCARDRAM ; TO MAIN
CF9F:      33 *
CF9F:      34 MOVESTRT EQU *
CF9F:B2 3C CF9F 35 MOVELOOP LDA (A1L) ;get a byte
CFA1:92 42 36 STA (A4L) ;move it
CFA3:E6 42 37 INC A4L
CFA5:D0 02 CFA9 38 BNE NEXTA1
CFA7:E6 43 39 INC A4H
CFA9:A5 3C 40 NEXTA1 LDA A1L
CFAB:C5 3E 41 CMP A2L
CFAD:A5 3D 42 LDA A1H
CFAF:E5 3F 43 SBC A2H
CFB1:E6 3C 44 INC A1L
CFB3:D0 02 CFB7 45 BNE CO1
CFB5:E6 3D 46 INC A1H
CFB7:90 E6 CF9F 47 CO1 BCC MOVELOOP ;=>more to move
CFB9:      48 *
CFB9:8D 04 CO 49 STA WRMAINRAM ;CLEAR FLAG2
CFBC:68 50 PLA ;GET ORIGINAL STATE
CFBD:10 03 CFC2 51 BPL CO3 ;=>IT WAS OFF
CFBF:8D 05 CO 52 STA WRCARDRAM
CFC2: CFC2 53 CO3 EQU *
CFC2:8D 02 CO 54 STA RDMAINRAM ;CLEAR FLAG1
CFC5:68 55 PLA ;GET ORIGINAL STATE
CFC6:10 03 CFCB 56 BPL MOVERET ;=>IT WAS OFF
CFC8:8D 03 CO 57 STA RDCARDRAM
CFCB: CFCB 58 MOVERET EQU *
CFCB:68 59 PLA ;Restore AC
CFCC:60 60 RTS

```

```

CFCD:      62 *****
CFCD:      63 * NAME      : XFER
CFCD:      64 * FUNCTION: TRANSFER CONTROL CROSSBANK
CFCD:      65 * INPUT   : $03ED-TRANSFER ADDR
CFCD:      66 *           : CARRY SET=XFER TO CARD
CFCD:      67 *           : CLR=XFER TO MAIN
CFCD:      68 *           : VFLAG CLR=USE STD ZP/STK
CFCD:      69 *           : SET=USE ALT ZP/STK
CFCD:      70 * OUTPUT  : NONE
CFCD:      71 * VOLATILE: $03ED/03EE IN DEST BANK
CFCD:      72 * CALLS   : NOTHING
CFCD:      73 * NOTE    : ENTERED VIA JMP, NOT JSR
CFCD:      74 *****
CFCD:      75 *
CFCD:      76 XFER      EQU      *
CFCD:48    CFCD      77 PNA                      ;SAVE AC ON CURRENT STACK
CFCE:      78 *
CFCE:      79 * COPY DESTINATION ADDRESS TO THE
CFCE:      80 * OTHER BANK SO THAT WE HAVE IT
CFCE:      81 * IN CASE WE DO A SWAP:
CFCE:      82 *
CFCE:AD ED 03    83 LDA      $03ED      ;GET XFERADDR LO
CFD1:48    84 PNA                      ;SAVE ON CURRENT STACK
CFD2:AD EE 03    85 LDA      $03EE      ;GET XFERADDR HI
CFD5:48    86 PNA                      ;SAVE IT TOO
CFD6:      87 *
CFD6:      88 * SWITCH TO APPROPRIATE BANK:
CFD6:      89 *
CFD6:90 08 CFE0   90 BCC      XFERC2M      ;=>CARD->MAIN
CFD8:8D 03 C0    91 STA      RDCARDRAM    ;SET FOR RUNNING
CFDB:8D 05 C0    92 STA      WRCARDRAM    ; IN CARD RAM
CFDE:B0 06 CFE6   93 BCS      XFERZP      ;=> always taken
CFE0:      CFE0   94 XFERC2M      EQU      *
CFE0:8D 02 C0    95 STA      RDMAINRAM    ;SET FOR RUNNING
CFE3:8D 04 C0    96 STA      WRMAINRAM    ; IN MAIN RAM
CFE6:      97 *
CFE6:      CFE6   98 XFERZP      EQU      *      ;SWITCH TO ALT ZP/STK
CFE6:68    99 PLA                      ;STUFF XFERADDR
CFE7:8D EE 03   100 STA      $03EE      ; HI AND
CFEA:68    101 PLA
CFE8:8D ED 03   102 STA      $03ED      ; LO
CFEE:68    103 PLA                      ;RESTORE AC
CFEF:70 05 CFF6   104 BVS      XFERAZP      ;=>switch in alternate zp
CFF1:8D 08 C0    105 STA      SETSTDZP      ;else force standard zp
CFF4:50 03 CFF9   106 BVC      JMPDEST      ;=>always perform transfer
CFF6:8D 09 C0    107 XFERAZP      STA      SETALTZP      ;switch in alternate zp
CFF9:6C ED 03   108 JMPDEST      JMP      ($03ED)      ;=>off we go
CFFC:      109 *****
CFFC:      0004   110 DS          $0000-*, $00
----- NEXT OBJECT FILE NAME IS FIRM.1
F800:      F800   31 ORG      F800
F800:      32 INCLUDE AUTOST1      ;F8 monitor rom

```

F800:4A	3	PLOT	LSR	A	;Y-COORD/2
F801:08	4		PHP		;SAVE LSB IN CARRY
F802:20 47 F8	5		JSR	GBASCALC	;CALC BASE ADR IN GBASL,H
F805:28	6		PLP		;RESTORE LSB FROM CARRY
F806:A9 0F	7		LDA	#\$0F	;MASK \$0F IF EVEN
F808:90 02 F80C	8		BCC	RTMASK	
F80A:69 E0	9		ADC	#\$E0	;MASK \$F0 IF ODD
F80C:85 2E	10	RTMASK	STA	MASK	
F80E:B1 26	11	PLOT1	LDA	(GBASL),Y	;DATA
F810:45 30	12		EOR	COLOR	; XOR COLOR
F812:25 2E	13		AND	MASK	; AND MASK
F814:51 26	14		EOR	(GBASL),Y	; XOR DATA
F816:91 26	15		STA	(GBASL),Y	; TO DATA
F818:60	16		RTS		
F819:	17	*			
F819:20 00 F8	18	HLINE	JSR	PLOT	;PLOT SQUARE
F81C:C4 2C	19	HLINE1	CPY	H2	;DONE?
F81E:B0 11 F831	20		BCS	RTS1	; YES, RETURN
F820:C8	21		INY		; NO, INCR INDEX (X-COORD)
F821:20 0E F8	22		JSR	PLOT1	;PLOT NEXT SQUARE
F824:90 F6 F81C	23		BCC	HLINE1	;ALWAYS TAKEN
F826:69 01	24	VLINEZ	ADC	#\$01	;NEXT Y-COORD
F828:48	25	VLINE	PHA		; SAVE ON STACK
F829:20 00 F8	26		JSR	PLOT	; PLOT SQUARE
F82C:68	27		PLA		
F82D:C5 2D	28		CMP	V2	;DONE?
F82F:90 F5 F826	29		BCC	VLINEZ	; NO, LOOP.
F831:60	30	RTS1	RTS		
F832:	31	*			
F832:A0 2F	32	CLRSCR	LDY	#\$2F	;MAX Y, FULL SCRNR CLR
F834:00 02 F838	33		BNE	CLRSC2	;ALWAYS TAKEN
F836:A0 27	34	CLRTOP	LDY	#\$27	;MAX Y, TOP SCRNR CLR
F838:84 2D	35	CLRSC2	STY	V2	;STORE AS BOTTOM COORD
F83A:	36	;			FOR VLINE CALLS
F83A:A0 27	37		LDY	#\$27	;RIGHTMOST X-COORD (COLUMN)
F83C:A9 00	38	CLRSC3	LDA	#\$00	;TOP COORD FOR VLINE CALLS
F83E:85 30	39		STA	COLOR	;CLEAR COLOR (BLACK)
F840:20 28 F8	40		JSR	VLINE	;DRAW VLINE
F843:88	41		DEY		;NEXT LEFTMOST X-COORD
F844:10 F6 F83C	42		BPL	CLRSC3	;LOOP UNTIL DONE.
F846:60	43		RTS		
F847:	44	*			
F847:48	45	GBASCALC	PHA		;FOR INPUT 00DEFCH
F848:4A	46		LSR	A	
F849:29 03	47		AND	#\$03	
F84B:09 04	48		ORA	#\$04	;GENERATE GBASH=000001FG
F84D:85 27	49		STA	GBASH	
F84F:68	50		PLA		;AND GBASL=HDEDE000
F850:29 18	51		AND	#\$18	
F852:90 02 F856	52		BCC	GBCALC	
F854:69 7F	53		ADC	#\$7F	
F856:85 26	54	GBCALC	STA	GBASL	
F858:0A	55		ASL	A	
F859:0A	56		ASL	A	
F85A:05 26	57		ORA	GBASL	
F85C:85 26	58		STA	GBASL	
F85E:60	59		RTS		
F85F:	60	*			



F85F:A5 30	61	NXTCOL	LDA	COLOR	;INCREMENT COLOR BY 3
F861:18	62		CLC		
F862:69 03	63		ADC	#\$03	
F864:29 0F	64	SETCOL	AND	#\$0F	;SETS COLOR=17*A MOD 16
F866:85 30	65		STA	COLOR	
F868:0A	66		ASL	A	;BOTH HALF BYTES OF COLOR EQUAL
F869:0A	67		ASL	A	
F86A:0A	68		ASL	A	
F86B:0A	69		ASL	A	
F86C:05 30	70		ORA	COLOR	
F86E:85 30	71		STA	COLOR	
F870:60	72		RTS		
F871:	73	*			
F871:4A	74	SCRN	LSR	A	;READ SCREEN Y-COORD/2
F872:08	75		PHP		;SAVE LSB (CARRY)
F873:20 47 F8	76		JSR	GBASCALC	;CALC BASE ADDRESS
F876:B1 26	77		LDA	(GBASL),Y	;GET BYTE
F878:28	78		PLP		;RESTORE LSB FROM CARRY
F879:90 04 F87F	79	SCRN2	BCC	RTMSKZ	;IF EVEN, USE LO H
F878:4A	80		LSR	A	
F87C:4A	81		LSR	A	
F87D:4A	82		LSR	A	;SHIFT HIGH HALF BYTE DOWN
F87E:4A	83		LSR	A	
F87F:29 0F	84	RTMSKZ	AND	#\$0F	;MASK 4-BITS
F881:60	85		RTS		
F882:	86	*			
F882:A6 3A	87	INSDS1	LDX	PCL	;PRINT PCL,H
F884:A4 3B	88		LDY	PCH	
F886:20 96 FD	89		JSR	PRYX2	
F889:20 48 F9	90		JSR	PRBLNK	;FOLLOWED BY A BLANK
F88C:A1 3A	91	INSDS2	LDA	(PCL,X)	;GET OPCODE
F88E:A8	92		TAY		
F88F:4A	93		LSR	A	;EVEN/ODD TEST
F890:90 05 F897	94		BCC	IEVEN	
F892:6A	95		ROR	A	;BIT 1 TEST
F893:B0 0C F8A1	96		BCS	ERR	;XXXXXX!! INVALID OP
F895:29 87	97		AND	#\$87	;MASK BITS
F897:4A	98	IEVEN	LSR	A	;LSB INTO CARRY FOR L/R TEST
F898:AA	99		TAX		
F899:BD 62 F9	100		LDA	FMT1,X	;GET FORMAT INDEX BYTE
F89C:20 79 F8	101		JSR	SCRN2	;R/L H-BYTE ON CARRY
F89F:D0 04 F8A5	102		BNE	GETFMT	
F8A1:A0 FC	103	ERR	LDY	#\$FC	;SUBSTITUTE \$FC FOR INVALID OPS
F8A3:A9 00	104		LDA	#\$00	;SET PRINT FORMAT INDEX TO 0
F8A5:AA	105	GETFMT	TAX		
F8A6:BD A6 F9	106		LDA	FMT2,X	;INDEX INTO PRINT FORMAT TABLE
F8A9:85 2E	107		STA	FORMAT	;SAVE FOR ADR FIELD FORMATTING
F8AB:29 03	108		AND	#\$03	;MASK FOR 2-BIT LENGTH
F8AD:	109	; (0=1 BYTE, 1=2 BYTE, 2=3 BYTE)			
F8AD:85 2F	110		STA	LENGTH	
F8AF:20 35 FC	111		JSR	NEWOPS	;get index for new opcodes
F8B2:F0 18 F8CC	112		BEQ	GOTONE	;found a new op (or no op)
F8B4:29 BF	113		AND	#\$8F	;MASK FOR 1XXX1010 TEST
F8B6:AA	114		TAX		;SAVE IT
F8B7:98	115		TYA		;OPCODE TO A AGAIN
F8B8:A0 03	116		LDY	#\$03	
F8BA:F0 8A	117		CPX	#\$8A	
F8BC:F0 0B F8C9	118		BEQ	MNNDX3	

F8BE:4A		119	MNNDX1	LSR	A	
F8BF:90 08	F8C9	120		BCC	MNNDX3	;FORM INDEX INTO MNEMONIC TABLE
F8C1:4A		121		LSR	A	
F8C2:4A		122	MNNDX2	LSR	A	; 1) 1XXX1010 => 00101XXX
F8C3:09 20		123		ORA	#\$20	; 2) XXXYYY01 => 00111XXX
F8C5:88		124		DEY		; 3) XXXYYY10 => 00110XXX
F8C6:D0 FA	F8C2	125		BNE	MNNDX2	; 4) XXXYY100 => 00100XXX
F8C8:C8		126		INY		; 5) XXXXX000 => 000XXXXX
F8C9:88		127	MNNDX3	DEY		
F8CA:D0 F2	F8BE	128		BNE	MNNDX1	
F8CC:60		129	GOTONE	RTS		
F8CD:		130	*			
F8CD:FF FF FF		131		DFB	\$FF,\$FF,\$FF	
F8D0:		132	*			
F8D0:20 82 F8		133	INSTDSP	JSR	INSDS1	;GEN FMT, LEN BYTES
F8D3:48		134		PHA		;SAVE MNEMONIC TABLE INDEX
F8D4:B1 3A		135	PRNTOP	LDA	(PCL),Y	
F8D6:20 DA FD		136		JSR	PRBYTE	
F8D9:A2 01		137		LDX	#\$01	;PRINT 2 BLANKS
F8DB:20 4A F9		138	PRNTBL	JSR	PRBL2	
F8DE:C4 2F		139		CPY	LENGTH	;PRINT INST (1-3 BYTES)
F8E0:C8		140		INY		;IN A 12 CHR FIELD
F8E1:90 F1	F8D4	141		BCC	PRNTOP	
F8E3:A2 03		142		LDX	#\$03	;CHAR COUNT FOR MNEMONIC INDEX
F8E5:C0 04		143		CPY	#\$04	
F8E7:90 F2	F8DB	144		BCC	PRNTBL	
F8E9:68		145		PLA		;RECOVER MNEMONIC INDEX
F8EA:A8		146		TAY		
F8EB:B9 C0 F9		147		LDA	MNEM1,Y	
F8EE:85 2C		148		STA	LMNEM	;FETCH 3-CHAR MNEMONIC
F8F0:B9 00 FA		149		LDA	MNEMR,Y	; (PACKED INTO 2-BYTES)
F8F3:85 2D		150		STA	RMNEM	
F8F5:A9 00		151	PRMN1	LDA	#\$00	
F8F7:A0 05		152		LDY	#\$05	
F8F9:06 2D		153	PRMN2	ASL	RMNEM	;SHIFT 5 BITS OF CHARACTER INTO A
F8FB:26 2C		154		ROL	LMNEM	
F8FD:2A		155		ROL	A	; (CLEARS CARRY)
F8FE:88		156		DEY		
F8FE:D0 F8	F8F9	157		BNE	PRMN2	
F901:69 BF		158		ADC	#\$BF	;ADD "?" OFFSET
F903:20 ED FD		159		JSR	COUT	;OUTPUT A CHAR OF MNEM
F906:CA		160		DEX		
F907:D0 EC	F8F5	161		BNE	PRMN1	
F909:20 48 F9		162		JSR	PRBLNK	;OUTPUT 3 BLANKS
F90C:A4 2F		163		LDY	LENGTH	
F90E:A2 06		164		LDX	#\$06	;CNT FOR 6 FORMAT BITS
F910:E0 03		165	PRADR1	CPX	#\$03	
F912:F0 1C	F930	166		BEQ	PRADR5	;IF X=3 THEN ADDR.
F914:06 2E		167	PRADR2	ASL	FORMAT	
F916:90 0E	F926	168		BCC	PRADR3	
F918:BD B9 F9		169		LDA	CHAR1-1,X	
F91B:20 ED FD		170		JSR	COUT	
F91E:BD B3 F9		171		LDA	CHAR2-1,X	
F921:F0 03	F926	172		BEQ	PRADR3	
F923:20 ED FD		173		JSR	COUT	
F926:CA		174	PRADR3	DEX		
F927:D0 E7	F910	175		BNE	PRADR1	
F929:60		176		RTS		

F92A:		177 *			
F92A:88		178	PRADR4	DEY	
F92B:30	E7	179		BMI	PRADR2
F92D:20	DA	180		JSR	PRBYTE
F930:A5	2E	181	PRADR5	LDA	FORMAT
F932:C9	E8	182		CMP	#SE8
F934:B1	3A	183		LDA	(PCL),Y
F936:90	F2	184		BCC	PRADR4
F938:20	56	185	RELADR	JSR	PCADJ3
F93B:AA		186		TAX	
F93C:E8		187		INX	
F93D:D0	01	188		BNE	PRNTYX
F93F:C8		189		INY	
F940:98		190	PRNTYX	TVA	
F941:20	DA	191	PRNTAX	JSR	PRBYTE
F944:8A		192	PRNTX	TXA	
F945:4C	DA	193		JMP	PRBYTE
F948:		194 *			
F948:A2	03	195	PRBLNK	LDX	#S03
F94A:A9	A0	196	PRBL2	LDA	#S40
F94C:20	ED	197	PRBL3	JSR	COUT
F94F:CA		198		DEX	
F950:D0	F8	199		BNE	PRBL2
F952:60		200		RTS	
F953:		201 *			
F953:38		202	PCADJ	SEC	
F954:A5	2F	203	PCADJ2	LDA	LENGTH
F956:A4	3B	204	PCADJ3	LDY	PCH
F958:AA		205		TAX	
F959:10	01	206		BPL	PCADJ4
F95B:88		207		DEY	
F95C:65	3A	208	PCADJ4	ADC	PCL
F95E:90	01	209		BCC	RIS2
F960:C8		210		INY	
F961:60		211	RIS2	RTS	
F962:		212 *			
F962:		213	; FMT1 BYTES: XXXXXXYO INSTRS		
F962:		214	; IF Y=0 THEN RIGHT HALF BYTE		
F962:		215	; IF Y=1 THEN LEFT HALF BYTE		
F962:		216	; (X=INDEX)		
F962:		217 *			
F962:0F		218	FMT1	DFB	\$0F
F963:22		219		DFB	\$22
F964:FF		220		DFB	\$FF
F965:33		221		DFB	\$33
F966:CB		222		DFB	\$CB
F967:62		223		DFB	\$62
F968:FF		224		DFB	\$FF
F969:73		225		DFB	\$73
F96A:03		226		DFB	\$03
F96B:22		227		DFB	\$22
F96C:FF		228		DFB	\$FF
F96D:33		229		DFB	\$33
F96E:CB		230		DFB	\$CB
F96F:66		231		DFB	\$66
F970:FF		232		DFB	\$FF
F971:77		233		DFB	\$77
F972:0F		234		DFB	\$0F

;HANDLE REL ADR MODE  
 ;SPECIAL (PRINT TARGET,  
 ; NOT OFFSET)  
 ;PCL,PCH+OFFSET+1 TO A,Y  
 ;+1 TO Y,X  
 ;OUTPUT TARGET ADR  
 ; OF BRANCH AND RETURN  
 ;BLANK COUNT  
 ;LOAD A SPACE  
 ;OUTPUT A BLANK  
 ;LOOP UNTIL COUNT=0  
 ;0=1 BYTE, 1=2 BYTE,  
 ; 2=3 BYTE  
 ;TEST DISPLACEMENT SIGN  
 ; (FOR REL BRANCH)  
 ;EXTEND NEG BY DECR PCH  
 ;PCL+LENGTH(OR DISPL)+1 TO A  
 ; CARRY INTO Y (PCH)

F973:20	235	DFB	\$20	
F974:FF	236	DFB	\$FF	
F975:33	237	DFB	\$33	
F976:CB	238	DFB	\$CB	
F977:60	239	DFB	\$60	
F978:FF	240	DFB	\$FF	
F979:70	241	DFB	\$70	
F97A:0F	242	DFB	\$0F	
F97B:22	243	DFB	\$22	
F97C:FF	244	DFB	\$FF	
F97D:39	245	DFB	\$39	
F97E:CB	246	DFB	\$CB	
F97F:66	247	DFB	\$66	
F980:FF	248	DFB	\$FF	
F981:7D	249	DFB	\$7D	
F982:0B	250	DFB	\$0B	
F983:22	251	DFB	\$22	
F984:FF	252	DFB	\$FF	
F985:33	253	DFB	\$33	
F986:CB	254	DFB	\$CB	
F987:A6	255	DFB	\$A6	
F988:FF	256	DFB	\$FF	
F989:73	257	DFB	\$73	
F98A:11	258	DFB	\$11	
F98B:22	259	DFB	\$22	
F98C:FF	260	DFB	\$FF	
F98D:33	261	DFB	\$33	
F98E:CB	262	DFB	\$CB	
F98F:A6	263	DFB	\$A6	
F990:FF	264	DFB	\$FF	
F991:87	265	DFB	\$87	
F992:01	266	DFB	\$01	
F993:22	267	DFB	\$22	
F994:FF	268	DFB	\$FF	
F995:33	269	DFB	\$33	
F996:CB	270	DFB	\$CB	
F997:60	271	DFB	\$60	
F998:FF	272	DFB	\$FF	
F999:70	273	DFB	\$70	
F99A:01	274	DFB	\$01	
F99B:22	275	DFB	\$22	
F99C:FF	276	DFB	\$FF	
F99D:33	277	DFB	\$33	
F99E:CB	278	DFB	\$CB	
F99F:60	279	DFB	\$60	
F9A0:FF	280	DFB	\$FF	
F9A1:70	281	DFB	\$70	
F9A2:24	282	DFB	\$24	
F9A3:31	283	DFB	\$31	
F9A4:65	284	DFB	\$65	
F9A5:78	285	DFB	\$78	
F9A6:	286	; ZZXXXY01 INSTR'S		
F9A6:00	287	FMT2	DFB	\$00 ;ERR
F9A7:21	288		DFB	\$21 ;1MM
F9A8:81	289		DFB	\$81 ;Z-PAGE
F9A9:82	290		DFB	\$82 ;ABS
F9AA:59	291		DFB	\$59 ;(ZPAG,X)
F9AB:4D	292		DFB	\$4D ;(ZPAG),Y

F9AC:91	293	DFB	\$91	;ZPAG,X
F9AD:92	294	DFB	\$92	;ABS,X
F9AE:86	295	DFB	\$86	;ABS,Y
F9AF:4A	296	DFB	\$4A	;(ABS)
F9B0:85	297	DFB	\$85	;ZPAG,Y
F9B1:9D	298	DFB	\$9D	;RELATIVE
F9B2:49	299	DFB	\$49	;(ZPAG) (new)
F9B3:5A	300	DFB	\$5A	;(ABS,X) (new)
F9B4:	301 *			
F9B4:D9	302 CHAR2	DFB	\$D9	;'Y'
F9B5:00	303	DFB	\$00	; (byte F of FMT2)
F9B6:D8	304	DFB	\$D8	;'Y'
F9B7:A4	305	DFB	\$A4	;'S'
F9B8:A4	306	DFB	\$A4	;'S'
F9B9:00	307	DFB	\$00	
F9BA:	308 *			
F9BA:AC	309 CHAR1	DFB	\$AC	;' '
F9BB:A9	310	DFB	\$A9	;' )'
F9BC:AC	311	DFB	\$AC	;' '
F9BD:A3	312	DFB	\$A3	;' #'
F9BE:A8	313	DFB	\$A8	;' ('
F9BF:A4	314	DFB	\$A4	;'S'
F9C0:1C	315 MNEML	DFB	\$1C	
F9C1:8A	316	DFB	\$8A	
F9C2:1C	317	DFB	\$1C	
F9C3:23	318	DFB	\$23	
F9C4:5D	319	DFB	\$5D	
F9C5:8B	320	DFB	\$8B	
F9C6:1B	321	DFB	\$1B	
F9C7:A1	322	DFB	\$A1	
F9C8:9D	323	DFB	\$9D	
F9C9:8A	324	DFB	\$8A	
F9CA:1D	325	DFB	\$1D	
F9CB:23	326	DFB	\$23	
F9CC:9D	327	DFB	\$9D	
F9CD:8B	328	DFB	\$8B	
F9CE:1D	329	DFB	\$1D	
F9CF:A1	330	DFB	\$A1	
F9D0:1C	331	DFB	\$1C	;BRA
F9D1:29	332	DFB	\$29	
F9D2:19	333	DFB	\$19	
F9D3:AE	334	DFB	\$AE	
F9D4:69	335	DFB	\$69	
F9D5:A8	336	DFB	\$A8	
F9D6:19	337	DFB	\$19	
F9D7:23	338	DFB	\$23	
F9D8:24	339	DFB	\$24	
F9D9:53	340	DFB	\$53	
F9DA:1B	341	DFB	\$1B	
F9DB:23	342	DFB	\$23	
F9DC:24	343	DFB	\$24	
F9DD:53	344	DFB	\$53	
F9DE:19	345	DFB	\$19	
F9DF:A1	346	DFB	\$A1	; (A) FORMAT ABOVE
F9E0:AD	347	DFB	\$AD	; TSB
F9E1:1A	348	DFB	\$1A	
F9E2:5B	349	DFB	\$5B	
F9E3:5B	350	DFB	\$5B	

F9E4:A5	351	DFB	\$A5	
F9E5:69	352	DFB	\$69	
F9E6:24	353	DFB	\$24	
F9E7:24	354	DFB	\$24	; (B) FORMAT
F9E8:AE	355	DFB	\$AE	
F9E9:AE	356	DFB	\$AE	
F9EA:A8	357	DFB	\$A8	
F9EB:AD	358	DFB	\$AD	
F9EC:29	359	DFB	\$29	
F9ED:8A	360	DFB	\$8A	
F9EE:7C	361	DFB	\$7C	
F9EF:8B	362	DFB	\$8B	; (C) FORMAT
F9F0:15	363	DFB	\$15	
F9F1:9C	364	DFB	\$9C	
F9F2:6D	365	DFB	\$6D	
F9F3:9C	366	DFB	\$9C	
F9F4:A5	367	DFB	\$A5	
F9F5:69	368	DFB	\$69	
F9F6:29	369	DFB	\$29	
F9F7:53	370	DFB	\$53	; (D) FORMAT
F9F8:84	371	DFB	\$84	
F9F9:13	372	DFB	\$13	
F9FA:34	373	DFB	\$34	
F9FB:11	374	DFB	\$11	
F9FC:A5	375	DFB	\$A5	
F9FD:69	376	DFB	\$69	
F9FE:23	377	DFB	\$23	; (E) FORMAT
F9FF:A0	378	DFB	\$A0	
FA00:	379 *			
FA00:D8	380 MNEMR	DFB	\$D8	
FA01:62	381	DFB	\$62	
FA02:5A	382	DFB	\$5A	
FA03:48	383	DFB	\$48	
FA04:26	384	DFB	\$26	
FA05:62	385	DFB	\$62	
FA06:94	386	DFB	\$94	
FA07:88	387	DFB	\$88	
FA08:54	388	DFB	\$54	
FA09:44	389	DFB	\$44	
FA0A:C8	390	DFB	\$C8	
FA0B:54	391	DFB	\$54	
FA0C:68	392	DFB	\$68	
FA0D:44	393	DFB	\$44	
FA0E:E8	394	DFB	\$E8	
FA0F:94	395	DFB	\$94	
FA10:C4	396	DFB	\$C4	; BRA
FA11:B4	397	DFB	\$B4	
FA12:08	398	DFB	\$08	
FA13:84	399	DFB	\$84	
FA14:74	400	DFB	\$74	
FA15:B4	401	DFB	\$B4	
FA16:28	402	DFB	\$28	
FA17:6E	403	DFB	\$6E	
FA18:74	404	DFB	\$74	
FA19:F4	405	DFB	\$F4	
FA1A:CC	406	DFB	\$CC	
FA1B:4A	407	DFB	\$4A	
FA1C:72	408	DFB	\$72	



FA1D:F2	409	DFB	\$F2	
FA1E:A4	410	DFB	\$A4	
FA1F:8A	411	DFB	\$8A	; (A) FORMAT
FA20:06	412	DFB	\$06	; TSB
FA21:AA	413	DFB	\$AA	
FA22:A2	414	DFB	\$A2	
FA23:A2	415	DFB	\$A2	
FA24:74	416	DFB	\$74	
FA25:74	417	DFB	\$74	
FA26:74	418	DFB	\$74	
FA27:72	419	DFB	\$72	; (B) FORMAT
FA28:44	420	DFB	\$44	
FA29:68	421	DFB	\$68	
FA2A:B2	422	DFB	\$B2	
FA2B:32	423	DFB	\$32	
FA2C:B2	424	DFB	\$B2	
FA2D:72	425	DFB	\$72	
FA2E:22	426	DFB	\$22	
FA2F:72	427	DFB	\$72	; (C) FORMAT
FA30:1A	428	DFB	\$1A	
FA31:1A	429	DFB	\$1A	
FA32:26	430	DFB	\$26	
FA33:26	431	DFB	\$26	
FA34:72	432	DFB	\$72	
FA35:72	433	DFB	\$72	
FA36:88	434	DFB	\$88	
FA37:C8	435	DFB	\$C8	; (D) FORMAT
FA38:C4	436	DFB	\$C4	
FA39:CA	437	DFB	\$CA	
FA3A:26	438	DFB	\$26	
FA3B:48	439	DFB	\$48	
FA3C:44	440	DFB	\$44	
FA3D:44	441	DFB	\$44	
FA3E:A2	442	DFB	\$A2	
FA3F:C8	443	DFB	\$C8	; (E) FORMAT
FA40:	444 *			
FA40:48	445 IRQ	PHA		;save accumulator
FA41:68	446	PLA		;rescued by stack trick later
FA42:68	447	PLA		
FA43:4C 06 C8	448	JMP	IRQ1	;do rest of IRQ handler
FA46:	449 *			
FA46:EA	450	NOP		
FA47:	451 *			
FA47:	452 *			
FA47:	453 *			
FA47:	454 *			
FA47:	455 *			
FA47:	456 *			
FA47:	457 *			
FA47:85 44	458 NEWBRK	STA	MACSTAT	;save state of machine
FA49:7A	459	PLY		;restore registers for save
FA4A:FA	460	PLX		
FA4B:68	461	PLA		
FA4C:	462 *			
FA4C:28	463 BREAK	PLP		;Note: same as old BREAK routine!!
FA4D:20 4A FF	464	JSR	SAVE	;save reg's on BRK
FA50:68	465	PLA		;including PC
FA51:85 3A	466	STA	PCL	

```

FA53:68          467          PLA
FA54:85 3B       468          STA    PCH
FA56:6C F0 03    469          JMP    (BRKV)      ;call BRX HANDLER
FA59:            470 *
FA59:20 82 F8    471 OLD BRK JSR    INSDSI      ;PRINT USER PC
FA5C:20 DA FA    472          JSR    RCDSP1      ; AND REGS
FA5F:4C 65 FF    473          JMP    MON        ;GO TO MONITOR (NO PASS GO, NO $200!)
FA62:            474 *
FA62:D8          475 RESET   CLD                ;DO THIS FIRST THIS TIME
FA63:20 84 FE    476          JSR    SETNORM
FA66:20 2F FB    477          JSR    INIT
FA69:20 93 FE    478          JSR    SETVID
FA6C:20 89 FE    479          JSR    SETKBD
FA6F:20 1C C4    480          JSR    INITMOUSE   ;initialize the mouse
FA72:20 04 CC    481          JSR    CLRPORT    ;clear port setup bytes
FA75:9C FF 04    482          SIZ    ACIABUF    ;and the commahead buffer
FA78:AD 5F C0    483          LDA    SETAN3      ; AN3 = TTL HI
FA7B:20 BD FA    484          JSR    RESET.X     ; initialize other devices
FA7E:2C 10 C0    485          BIT    KBDSTRB    ; CLEAR KEYBOARD
FA81:D8          486 NEWMON   CLD
FA82:20 3A FF    487          JSR    BELL        ; CAUSES DELAY IF KEY BOUNCES
FA85:AD F3 03    488          LDA    SOFTEV+1    ;IS RESET HI
FA88:49 A5       489          EOR    #A5        ;A FUNNY COMPLEMENT OF THE
FA8A:CD F4 03    490          CMP    PWRDUP     ; PWR UP BYTE ???
FA8D:D0 17 FAA6 491          BNE    PWRUP      ; NO SO PWRUP
FA8F:AD F2 03    492          LDA    SOFTEV     ; YES SEE IF COLD START
FA92:D0 3B FAF6 493          BNE    NOFIX     ; HAS BEEN DONE YET?
FA94:A9 E0       494          LDA    #E0        ; DOES SEV POINT AT BASIC?
FA96:CD F3 03    495          CMP    SOFTEV+1
FA99:D0 34 FAF6 496          BNE    NOFIX     ; YES SO REENTER SYSTEM
FA9B:A0 03       497 FIXSEV   LDY    #3        ; NO SO POINT AT WARM START
FA9D:8C F2 03    498          STY    SOFTEV     ; FOR NEXT RESET
FAA0:4C 00 E0    499          JMP    BASIC      ; AND DO THE COLD START
FAA3:            500 *
FAA3:20 3A FF    501 BEEPFIX JSR    BELL        ;Beep on powerup
FAA6:            502 *
FAA6:20 CA FC    503 PWRUP   JSR    COLDSTART   ;Trash memory, init ports
FAA9:            504 SETPG3   EQU    *          ; SET PAGE 3 VECTORS
FAA9:A2 05       505          LDX    #5
FAAB:BD FC FA    506 SETPLP   LDA    PWRCON-1,X  ; WITH CNTRL B ADRS
FAAE:9D EF 03    507          STA    BRKV-1,X     ; OF CURRENT BASIC
FAB1:CA          508          DEX
FAB2:D0 F7 FAAB 509          BNE    SETPLP
FAB4:A9 C6       510          LDA    #C6        ; LOAD HI SLOT +1
FAB6:80 5A FB12 511          BRA    PWRUP2    ;branch around mnemonics
FAB8:            512 *
FAB8:            513 * Extension to MNEML (left mnemonics)
FAB8:            514 *
FAB8:8A          515          DFB    $8A        ;PHY
FAB9:8B          516          DFB    $8B        ;PLY
FABA:A5          517          DFB    $A5        ;STZ
FABB:AC          518          DFB    $AC        ;TRB
FABC:00          519          DFB    $00        ;???
FABD:            520 *
FABD:            521 * This extension to the monitor reset routine ($FA62)
FABD:            522 * checks for apple keys. If both are pressed, it goes
FABD:            523 * into an exerciser mode. If the open apple key only is
FABD:            524 * pressed, memory is selectively trashed and a cold start

```

```

FABD:          525 * is done.
FABD:          526 *
FABD:A9 FF     527 RESET.X   LDA   #$FF
FABF:8D FB 04   528         STA   VMODE      ;initialize mode
FAC2:0E 62 C0   529         ASL   BUTN1
FAC5:2C 61 C0   530         BIT   BUTN0
FAC8:10 64 FB2E 531         BPL   RTS2D
FACA:90 D7 FAA3 532         BCC   BEEPPIX      ;open apple only, reboot
FACC:4C 7C C7   533         JMP   BANGER      ;both apples, exercise 'er
FACF:          534 *
FACF:6C F2 03   535 NOFIX    JMP   (SOFTV)
FAD2:          536 *
FAD2:C1 D8 D9 D0 537 RTBL    ASC   'AXYPS'
FAD7:          538 *
FAD7:20 8E FD   539 RGDSP    JSR   CROUT      ;DISPLAY USER REG CONTENTS
FADA:A9 45      540 RGDSP1   LDA   #$45      ;WITH LABELS
FADC:85 40      541         STA   A3L
FADE:A9 00      542         LDA   #$00
FAE0:85 41      543         STA   A3H
FAE2:A2 FB      544         LDX   #$FB
FAE4:A9 A0      545 RDSP1    LDA   #$A0
FAE6:20 ED FD   546         JSR   COUT
FAE9:BD D7 F9   547         LDA   RTBL-251,X
FAEC:20 ED FD   548         JSR   COUT
FAEF:A9 BD      549         LDA   #$BD
FAF1:20 ED FD   550         JSR   COUT
FAF4:85 4A      551         LDA   ACC+5,X
FAF6:80 0A FB02 552         BRA   RGDSP2      ;make room for mnemonics
FAF8:          553 *
FAF8:          554 * Right half of new mnemonics, indexed from MNEMR
FAF8:          555 *
FAF8:74         556         DFB   $74      ;PHY
FAF9:74         557         DFB   $74      ;PLY
FAFA:76         558         DFB   $76      ;STZ
FAFB:C6         559         DFB   $C6      ;TRB
FAFC:00         560         DFB   $00      ;???
FAFD:          561 *
FAFD:59 FA      562 PWRCON   DW    OLDBRK
FAFF:00 E0 45   563         DFB   $00,$E0,$45
FB02:          564 *
FB02:20 DA FD   565 RGDSP2   JSR   PRBYTE
FB05:E8         566         INX
FB06:30 DC FAE4 567         BMI   RDSP1
FB08:60         568         RTS
FB09:          569 *
FB09:C1 F0 F0 EC 570 TITLE   ASC   'Apple   ||'
FB11:C4         571         DFB   $C4      ;optional filler
FB12:          572 *
FB12:86 00      573 PWRUP2   STX   LOCO      ; SETPG3 MUST RETURN X=0
FB14:85 01      574         STA   LOC1      ; SET PTR H
FB16:20 60 FB   575         JSR   APPLRTI    ;Display our banner...
FB19:6C 00 00   576         JMP   (LOCO)    ;JUMP $C600
FB1C:00         577         BRK
FB1D:00         578         BRK
FB1E:          579 *
FB1E:4C DE C7   580 PREAD    JMP   MPADDLE    ;read mouse paddle
FB21:A0 00      581         LDY   #$00      ;INIT COUNT
FB23:EA         582         NOP      ;COMPENSATE FOR 1ST COUNT

```

FB24:EA		583	NOP		
FB25:BD	64 CO	584	PREAD2	LDA	PADDL0,X ;COUNT Y-REG EVERY 12 USEC.
FB28:10	04 FB2E	585		BPL	RTS2D
FB2A:C8		586		INY	
FB2B:D0	F8 FB25	587		BNE	PREAD2 ;EXIT AT 255 MAX
FB2D:88		588		DEY	
FB2E:60		589	RTS2D	RTS	
FB2F:		33		INCLUDE	AUTOST2

```

FB2F:      2 *
FB2F:A9 00  3 INIT      LDA    #$00          ;CLR STATUS FOR DEBUG SOFTWARE
FB31:85 48  4          STA    STATUS
FB33:AD 56 C0 5          LDA    LORES
FB36:AD 54 C0 6          LDA    TXTPAGE1      ;INIT VIDEO MODE
FB39:AD 51 C0 7 SETTXT   LDA    TXTSET       ;SET FOR TEXT MODE
FB3C:A9 00  8          LDA    #$00          ;FULL SCREEN WINDOW
FB3E:FO 08 FB4B 9        BEQ    SETWND
FB40:AD 50 C0 10 SETCR   LDA    TXTCLR       ;SET FOR GRAPHICS MODE
FB43:AD 53 C0 11          LDA    MIXSET      ;LOWER 4 LINES AS TEXT WINDOW
FB46:20 36 FB 12          JSR    CLRTOP
FB49:A9 14 13          LDA    #$14
FB4B:85 22 14 SETWND   STA    WNDTOP      ;SET WINDOW
FB4D:EA 15          NOP
FB4E:EA 16          NOP
FB4F:20 0A CE 17        JSR    WNDREST     ;40/80 column width
FB52:80 05 FB59 18      BRA    VTAB23
FB54:      19 *
FB54:09 80 20 DOCTL    ORA    #$80          ;controls need high bit
FB56:4C 54 CD 21      JMP    CTLCHAR0     ;execute control char
FB59:      22 *
FB59:A9 17 23 VTAB23   LDA    #$17          ;VTAB TO ROW 23
FB5B:85 25 24 TABV    STA    CV           ;VTABS TO ROW IN A-REG
FB5D:4C 22 FC 25      JMP    VTAB        ;don't set OURCV!!
FB60:      26 *
FB60:20 58 FC 27 APPLEII JSR    HOME        ;CLEAR THE SCRIN
FB63:A0 09 28          LDY    #9
FB65:B9 02 FD 29 STITLE LDA    APPLE2C-1,Y  ;GET A CHAR
FB68:99 0D 04 30      STA    LINE1+13,Y    ;PUT IT AT TOP CENTER OF SCREEN
FB6B:88 31          DEY
FB6C:D0 F7 FB65 32     BNE    STITLE
FB6E:60 33          RTS
FB6F:      34 *
FB6F:AD F3 03 35 SETPWRC LDA    SOFTEV+1    ;ROUTINE TO CALCULATE THE 'FUNNY
FB72:49 A5 36          EOR    #$A5          ;COMPLEMENT' FOR THE RESET VECTOR
FB74:8D F4 03 37      STA    PWREDUP
FB77:60 38          RTS
FB78:      39 *
FB78:      40 VIDWAIT EQU    *              ;CHECK FOR A PAUSE (CONTROL-S).
FB78:C9 8D 41          CMP    #$8D          ;ONLY WHEN I HAVE A CR
FB7A:D0 18 FB94 42     BNE    NOWAIT        ;NOT SO, DO REGULAR
FB7C:AC 00 C0 43      LDY    KBD           ;IS KEY PRESSED?
FB7F:10 13 FB94 44     BPL    NOWAIT        ;NO.
FB81:C0 93 45          CPY    #$93          ;YES -- IS IT CTRL-S?
FB83:D0 0F FB94 46     RNE    NOWAIT        ;NOPE - IGNORE
FB85:2C 10 C0 47      BIT    KBDSTRB       ;CLEAR STROBE
FB88:AC 00 C0 48 KBDWAIT LDY    KBD        ;WAIT TILL NEXT KEY TO RESUME
FB8B:10 FB FB88 49     BPL    KBDWAIT      ;WAIT FOR KEYPRESS
FB8D:C0 83 50          CPY    #$83          ;IS IT CONTROL-C?
FB8F:FO 03 FB94 51     BEQ    NOWAIT        ;YES, SO LEAVE IT
FB91:2C 10 C0 52      BIT    KBDSTRB       ;CLR STROBE
FB94:2C 7B 06 53 NOWAIT BIT    VFACTV      ;is video firmware active?
FB97:30 64 FBFD 54     BMI    VIDOUT      ;=>no, do normal 40 column
FB99:89 60 55          BIT    #$60          ;is it a control?
FB9B:FO 87 FB54 56     BEQ    DOCTL        ;=>yes, do it
FB9D:20 B8 C3 57      JSR    STORCH        ;print w/inverse mask
FBA0:EE 7B 05 58 NEWADV INC    OURCH       ;advance cursor
FBA3:AD 7B 05 59      LDA    OURCH        ;and update others

```

FBA6:2C 1F C0	50	BIT	RD80VID	;but only if not 80 columns
FBA9:30 05 FBEO	61	BMI	NEWADV1	;=>80 columns, leav'em
FBA8:8D 78 04	62	STA	OLDCH	
FBAE:85 24	63	STA	CH	
FBB0:80 46 FBF8	64	NEWADV1	BRA	ADV2
FBB2:	65	*		;check for CR
FBB2:EA	66	NOP		
FBB3:	67	*		
FBB3:06	68	F8VERSION	DFB	GOODF8
FBB4:	69	*		;//e, chels ID byte
FBB4:10 06 FBBC	70	DOCOUT1	BPL	DCX
FBB6:C9 A0	71	CMF	#SA0	;=>video firmware active, no mask
FBB8:90 02 FBBC	72	BCC	DCX	;is it control char?
FBBA:25 32	73	AND	INVFLG	;=>yes, no mask
FBBC:4C F6 FD	74	DCX	JMP	COUTZ
FBBF:00	75	BRK		;else apply inverse mask
FBC0:	76	*		;and print character
FBC0:00	77		DFB	\$00
FBC1:	78	*		;chels ID byte
FBC1:48	79	BASCALC	PHA	
FBC2:4A	80	LSR	A	;CALC BASE ADDR IN BASL,H
FBC3:29 03	81	AND	#S03	;FOR GIVEN LINE NO.
FBC5:09 04	82	ORA	#S04	; O<=LINE NO.<=\$17
FBC7:85 29	83	STA	BASH	;ARG=00DABCDE, GENERATE
FBC9:68	84	PLA		; BASH=000001CD
FBCA:29 18	85	AND	#S18	; AND
FBC0:90 02 FBDO	86	BCC	BASCLC2	; BASL=EABAB000
FBC6:69 7F	87	ADC	#S7F	
FBD0:85 28	88	BASCLC2	STA	BASL
FBD2:0A	89	ASL	A	
FBD3:0A	90	ASL	A	
FBD4:05 28	91	ORA	BASL	
FBD6:85 28	92	STA	BASL	
FBD8:60	93	RTS		
FBD9:	94	*		
FBD9:C9 87	95	CHKBELL	CMF	#S87
FBD8:D0 12 FBDF	96	BNE	RTS2B	;BELL CHAR? (CONTROL-G)
FBD9:A9 40	97	BELL1	LDA	#S40
FBD9:20 A8 FC	98	JSR	WAIT	; NO, RETURN.
FBE2:A0 C0	99	LDY	#S0C	; YES...
FBE4:A9 0C	100	BELL2	LDA	#S0C
FBE6:20 A8 FC	101	JSR	WAIT	;TOGGLE SPEAKER AT 1 KHZ
FBE9:AD 30 C0	102	LDA	SPKR	; FOR .1 SEC.
FBE0:88	103	DEY		
FBE0:00 F5 FBE4	104	BNE	BELL2	
FBEF:60	105	RTS2B	RTS	
FBF0:	106	*		
FBF0:A4 24	107	STORADV	LDY	CH
FBF2:91 28	108	STA	(BASL),Y	;get 40 column position
FBF4:E6 24	109	ADVANCE	INC	CH
FBF6:A5 24	110	LDA	CH	;and store
FBF8:C5 21	111	ADV2	CMF	WNDWDTH
FBF8:B0 66 FC62	112	SCS	CR	;increment cursor
FBF0:60	113	RTS3	RTS	;BEYOND WINDOW WIDTH?
FBF0:	114	*		; YES, CR TO NEXT LINE.
FBF0:C9 A0	115	VIDOUT	CMF	#SA0
FBF0:B0 EF FBF0	116	BCS	STORADV	;CONTROL CHAR?
FC01:A8	117	TAY		; NO, OUTPUT IT.
				;INVERSE VIDEO?



FC02:10 EC	FBF0	118	BPL	STORADV	; YES, OUTPUT IT.
FC04:C9 8D		119	CMP	#\$8D	;CR?
FC06:F0 6B	FC73	120	BEQ	NEWCR	;Yes, use new routine
FC08:C9 8A		121	CMP	#\$8A	;LINE PEED?
FC0A:F0 5A	FC66	122	BEQ	LF	; IF SO, DO IT.
FC0C:C9 88		123	CMP	#\$88	;BACK SPACE? (CONTROL-H)
FC0E:D0 C9	FBD9	124	BNE	CHKBELL	; NO, CHECK FOR BELL.
FC10:20 E2	FE	125	JSR	DECCH	;decrement all cursor H indices
FC13:10 E7	FBFC	126	BPL	RTS3	;IF POSITIVE, OK; ELSE MOVE UP.
FC15:A5 21		127	LDA	WNDWDTH	;get window width,
FC17:20 EB	FE	128	JSR	WDTHCH	;and set CH's to WNDWDTH-1
FC1A:A5 22		129	LDA	WNDTOP	;CURSOR V INDEX
FC1C:C5 25		130	CMP	CV	
FC1E:B0 DC	FBFC	131	BCS	RTS3	;top line, exit
FC20:C6 25		132	DEC	CV	;not top, go up one
FC22:		133	*		
FC22:80 62	FC86	134	BRA	NEWVTAB	;go update OURCV
FC24:20 C1	FB	135	JSR	BASCALC	;calculate the base address
FC27:A5 20		136	LDA	WNDLFT	;get the left window edge
FC29:2C 1F	C0	137	BIT	RD80VID	;80 columns?
FC2C:10 02	FC30	138	BPL	VTAB40	;=>no, left edge ok
FC2E:4A		139	LSR	A	;divide width by 2
FC2F:18		140	CLC		;prepare to add
FC30:65 28		141	ADC	BASL	;add width to base
FC32:85 28		142	STA	BASL	
FC34:60		143	RTS4	RTS	
FC35:		144	*		
FC35:		145	*	NEWOPS translates the opcode in the Y register	
FC35:		146	*	to a mnemonic table index and returns with Z=1.	
FC35:		147	*	If Y is not a new opcode, Z=0.	
FC35:		148	*		
FC35:98		149	NEWOPS	TYA	;get the opcode
FC36:A2 16		150	LDX	#NUMOPS	;check through new opcodes
FC38:DD FE	FE	151	NEWOP1	CMP	OPTBL,X
FC3B:F0 43	FC80	152	BEQ	GETINDX	;=>yes, get new index
FC3D:CA		153	DEX		
FC3E:10 F8	FC38	154	BPL	NEWOP1	;else check next one
FC40:60		155	RTS		;not found, exit with BNE
FC41:		156	*		
FC41:00		157	BRK		
FC42:		158	*		
FC42:80 19	FC5D	159	CLREOP	BRA	CLREOP1
FC44:A5 25		160	CLREOP2	LDA	CV
FC46:48		161	CLEOP1	PHA	;SAVE CURRENT LINE NO. ON STACK
FC47:20 24	FC	162	JSR	VTABZ	;CALC BASE ADDRESS
FC4A:20 9E	FC	163	JSR	CLEOLZ	;CLEAR TO EOL. (SETS CARRY)
FC4D:A0 00		164	LDY	#\$00	;CLEAR FROM H INDEX=0 FOR REST
FC4F:68		165	PLA		;INCREMENT CURRENT LINE NO.
FC50:1A		166	INC	A	
FC51:C5 23		167	CMP	WNBSTM	;DONE TO BOTTOM OF WINDOW?
FC53:90 F1	FC46	168	BCC	CLEOP1	; NO, KEEP CLEARING LINES.
FC55:B0 CB	FC22	169	BCS	VTAB	; YES, TAB TO CURRENT LINE
FC57:00		170	BRK		
FC58:		171	*		
FC58:20 A5	CD	172	HOME	JSR	HOMECLR
FC5B:80 E7	FC44	173	BRA	CLREOP2	;then clear to end of page
FC5D:		174	*		
FC5D:20 9D	QC	175	CLREOP1	JSR	GETCUR
					;load Y with proper CH

FC60:80 E2	FC44	176	BRA	CLREOP2	;before clearing page
FC62:		177 *			
FC62:80 0F	FC73	178 CR	BRA	NEWCR	;only LF if not Pascal
FC64:00		179	BRK		
FC65:00		180	BRK		
FC66:		181 *			
FC66:E6 25		182 LF	INC	CV	;INCR CURSOR V. (DOWN 1 LINE)
FC68:A5 25		183	LDA	CV	
FC6A:C5 23		184	CMP	WNBDM	;OFF SCREEN?
FC6C:90 1A	FC88	185	BCC	NEWVTABZ	;set base+WNDLFT
FC6E:C6 25		186	DEC	CV	;DECR CURSOR V. (BACK TO BOTTOM)
FC70:		187 *			
FC70:4C 35 CB		188 SCROLL	JMP	SCROLLUP	;scroll the screen
FC73:		189 *			
FC73:20 E9 FE		190 NEWCR	JSR	CLRCH	;set CH's to 0
FC76:2C FB 04		191	BIT	VMODE	;is it Pascal?
FC79:10 0A	FC85	192	BPL	CRRTS	;pascal, no LF
FC7B:20 44 FD		193	JSR	NOESCAPE	;else clear escape mode
FC7E:80 E6	FC66	194	BRA	LF	;then do LF
FC80:		195 *			
FC80:BD 15 FF		196 GETINDX	LDA	INDX,X	;lookup index for mnemonic
FC83:A0 00		197	LDY	#0	;exit with BEQ
FC85:60		198 CRRTS	RTS		
FC86:		199 *			
FC86:A5 25		200 NEWVTAB	LDA	CV	;update //e CV
FC88:8D FB 05		201 NEWVTABZ	STA	OURCV	
FC8B:80 97	FC24	202	BRA	VTABZ	;and calc base+WNDLFT
FC8D:		203 *			
FC8D:20 9D CC		204 NEWCLREOL	JSR	GETCUR	;get current cursor
FC90:A9 A0		205 NEWCLEOLZ	LDA	#\$A0	;get a blank
FC92:2C 78 06		206	BIT	VFACIV	;if video firmware active,
FC95:30 02	FC99	207	BMI	NEWCI	;=>don't use inverse mask
FC97:25 32		208	AND	INVFLG	
FC99:4C C2 CB		209 NEWCI	JMP	DOCLR	;go do clear
FC9C:		210 *			
FC9C:80 EF	FC8D	211 CLREOL	BRA	NEWCLREOL	;get cursor and clear
FC9E:80 F0	FC90	212 CLEOLZ	BRA	NEWCLEOLZ	;clear from Y
FCA0:		213 *			
FCA0:A0 00		214 CLRLIN	LDY	#0	;clear entire line
FCA2:80 EC	FC90	215	BRA	NEWCLEOLZ	
FCA4:		216 *			
FCA4:7C 2A CD		217 CTLDO	JMP	(CTLADR,X)	;jump to proper routine
FCA7:		218 *			
FCA7:EA		219	NOP		
FCA8:		220 *			
FCA8:38		221 WAIT	SEC		
FCA9:48		222 WAIT2	PHA		
FCAA:E9 01		223 WAIT3	SBC	#\$01	
FCAC:D0 FC	FCAA	224	BNE	WAIT3	;1.0204 USEC
FCAE:68		225	PLA		; (13+2712*A+\$12*A*A)
FCAF:E9 01		226	SBC	#\$01	
FCB1:D0 F6	FCA9	227	BNE	WAIT2	
FCB3:60		228 RIS6	RTS		
FCB4:		229 *			
FCB4:E6 42		230 NXTA4	INC	A4L	;INCR 2-BYTE A4
FCB6:D0 02	FCBA	231	BNE	NXTA1	; AND A1
FCB8:E6 43		232	INC	A4H	
FCBA:A5 3C		233 NXTA1	LDA	A1L	;INCR 2-BYTE A1.

FCBC:C5 3E	234	CMP	A2L	; AND COMPARE TO A2
FCBE:A5 3D	235	LDA	AIH	; (CARRY SET IF >=)
FCC0:E5 3F	236	SBC	A2H	
FCC2:E6 3C	237	INC	A1L	
FCC4:D0 02 FCC8	238	BNE	RTS4B	
FCC6:E6 3D	239	INC	AIH	
FCC8:60	240	RTS		
FCC9:	241 *			
FCC9:60	242 HEADR	RTS		;don't do it
FCCA:	243 *			
FCCA:A0 B0	244 COLDSTART	LDY	#\$B0	;let it preprocess down
FCCC:64 3C	245	STZ	A1L	
FCCF:A2 BF	246	LDX	#\$BF	;start from BFXX down
FCD0:86 3D	247 BLAST	STX	AIH	
FCD2:A9 A0	248	LDA	#\$A0	;store blanks
FCD4:91 3C	249	STA	(A1L),Y	
FCD6:88	250	DEY		
FCD7:91 3C	251	STA	(A1L),Y	
FCD9:CA	252	DEX		;back down to next page
FCD:AEO 01	253	CPX	#1	;stay away from stack
FCD:C:D0 F2 FCD0	254	BNE	BLAST	;fall into COMINIT
FCDE:	255 *			
FCDE:8D 01 C0	256	STA	SET80COL	;init ALT screen holes
FCE1:AD 55 C0	257	LDA	TXTPAGE2	;for serial and comm ports
FCE4:38	258	SEC		
FCE5:A2 88	259	LDX	#\$88	
FCE7:BD 27 CB	260 COM1	LDA	COMTBL-1,X	;XFER from rom
FCEA:90 0A FCF6	261	BCC	COM2	;branch if defaults ok
FCEC:DD 77 04	262	CMP	\$477,X	;test for prior setup
FCEF:18	263	CLC		;branch if not valid
FCF0:D0 04 FCF6	264	BNE	COM2	;If \$4F8 & \$4FF = TBL values
FCF2:E0 82	265	CPX	#\$82	
FCF4:90 06 FCF6	266	BCC	COM3	
FCF6:9D 77 04	267 COM2	STA	\$477,X	
FCF9:CA	268	DEX		;move all 8...
FCFA:D0 EB FCE7	269	BNE	COM1	
FCFC:AD 54 C0	270 COM3	LDA	TXTPAGE1	;restore switches
FCFF:8D 00 C0	271	STA	CLR80COL	;to default states
FDO2:60	272	RTS		
FDO3:	273 *			
FDO3:	274	MSB	ON	
FDO3:C1 F0 F0 EC	275 APPLE2C	ASC	"Apple	//c"
FDOC:	276 *			
FDOC:A4 24	277 RDKEY	LDY	CH	;get char at current position
FDOE:B1 28	278	LDA	(BASL),Y	;for those who restore it
FD10:EA	279	NOP		;if a program controls input
FD11:EA	280	NOP		;hooks, no cursor may be displayed
FD12:EA	281	NOP		
FD13:EA	282	NOP		
FD14:EA	283	NOP		
FD15:EA	284	NOP		
FD16:EA	285	NOP		
FD17:EA	286	NOP		
FD18:	287 *			
FD18:6C 38 00	288 KEYIND	JMP	(KSWL)	;GO TO USER KEY-IN
FD1B:	289 *			
FD1B:91 28	290 KEYIN	STA	(BASL),Y	;erase false images
FD1D:20 4C CC	291	JSR	SHOWCUR	;display true cursor

FD20:20 70 CC	292	DONXTCUR	JSR	UPDATE	;look for key, blink II cursor
FD23:10 FB FD20	293		BPL	DONXTCUR	;loop until keypress
FD25:48	294	GOTKEY	FHA		;save character
FD26:A9 08	295		LDA	#M.CTL	;were escapes enabled?
FD28:2C FB 04	296		BIT	VMODE	
FD2B:D0 1D FD4A	297		BNE	NOESC2	;=>no, there is no escape
FD2D:68	298		PLA		;yes, there may be a way out!!
FD2E:C9 9B	299		CMF	#ESC	;escape?
FD30:D0 06 FD38	300		BNE	LOOKPICK	;=>no escape
FD32:4C CC CC	301		JMP	NEWESC	;=>go do escape sequence
FD35:	302	*			
FD35:4C ED CC	303	RDCHAR	JMP	ESCRDKEY	;do RDKEY with escapes
FD38:	304	*			
FD38:2C 78 06	305	LOOKPICK	BIT	VFACTV	;only process f.arrow
FD3B:30 07 FD44	306		BMI	NOESCAPE	;if video firmware is active
FD3D:C9 95	307		CMF	#PICK	;was it PICK? (->,CTL-U)
FD3F:D0 03 FD44	308		BNE	NOESCAPE	;no, just return
FD41:20 1D CC	309		JSR	PICKY	;yes, pick the character
FD44:	310	*			
FD44:	311	* NOESCAPE is used by GETCOUT too.			
FD44:	312	*			
FD44:48	313	NOESCAPE	PHA		;save it
FD45:A9 08	314	NOESC1	LDA	#M.CTL	;disable escape sequences
FD47:0C FB 04	315		TSB	VMODE	;and enable controls
FD4A:68	316	NOESC2	PLA		;by setting M.CTL
FD4B:60	317		RTS		
FD4C:	318	*			
FD4C:EA	319		NOP		
FD4D:	320	*			
FD4D:20 A6 C3	321	NOTCR	JSR	GETCOUT	;disable controls and print
FD50:C9 88	322		CMF	#\$88	;CHECK FOR EDIT KEYS
FD52:F0 1D FD71	323		BEQ	BCKSPC	; - BACKSPACE
FD54:C9 98	324		CMF	#\$98	
FD56:F0 0A FD62	325		BEQ	CANCEL	; - CONTROL-X
FD58:E0 F8	326		CPX	#\$F8	
FD5A:90 03 FD5F	327		BCC	NOTCR1	;MARGIN?
FD5C:20 3A FF	328		JSR	BELL	; YES, SOUND BELL
FD5F:E8	329	NOTCR1	INX		;ADVANCE INPUT INDEX
FD60:D0 13 FD75	330		BNE	NXTCHAR	
FD62:A9 DC	331	CANCEL	LDA	#\$DC	;BACKSLASH AFTER CANCELLED LINE
FD64:20 A6 C3	332		JSR	GETCOUT	
FD67:20 8E FD	333	GETLNZ	JSR	CROUT	;OUTPUT 'CR'
FD6A:A5 33	334	GEILN	LDA	PROMPT	;OUTPUT PROMPT CHAR
FD6C:20 ED FD	335		JSR	COUT	
FD6F:A2 01	336	GETLN1	LDX	#\$01	;INIT INPUT INDEX
FD71:8A	337	BCKSPC	TXA		
FD72:F0 F3 FD67	338		BEQ	GETLNZ	;WILL BACKSPACE TO 0
FD74:CA	339		DEX		
FD75:20 ED CC	340	NXTCHAR	JSR	ESCRDKEY	;do new RDCHAR (allow escapes)
FD7B:C9 95	341		CMF	#PICK	;USE SCREEN CHAR
FD7A:D0 08 FD84	342		BNE	ADDINP	; FOR CONTROL-U
FD7C:20 1D CC	343		JSR	PICKY	;lift char from screen
FD7F:EA	344		NOP		
FD80:EA	345		NOP		
FD81:EA	346		NOP		;no upshifting needed
FD82:EA	347		NOP		
FD83:EA	348		NOP		
FD84:9D 00 02	349	ADDINP	STA	IN,X	;ADD TO INPUT BUFFER

FD87:C9 8D	350	CMF	#\$8D	
FD89:D0 C2 FD4D	351	BNE	NOTCR	
FD8B:20 9C FC	352 CROUT1	JSR	CLREOL	;CLR TO EOL IF CR
FD8E:A9 8D	353 CROUT	LDA	#\$8D	
FD90:D0 5B FDED	354	BNE	COUT	; (ALWAYS)
FD92:	355 *			
FD92:A4 3D	356 PRA1	LDY	A1H	;PRINT CR,A1 IN HEX
FD94:A6 3C	357	LDX	A1L	
FD96:20 8E FD	358 PRYX2	JSR	CROUT	
FD99:20 40 F9	359	JSR	PRNTYX	
FD9C:A0 00	360	LDY	#\$00	
FD9E:A9 AD	361	LDA	#\$AD	;PRINT '-'
FDA0:4C ED FD	362	JMP	COUT	
FDA3:	363 *			
FDA3:A5 3C	364 XAMB	LDA	A1L	
FDA5:09 07	365	ORA	#\$07	;SET TO FINISH AT
FDA7:85 3E	366	STA	A2L	; MOD 8=7
FDA9:A5 3D	367	LDA	A1H	
FDAB:85 3F	368	STA	A2H	
FDAD:A5 3C	369 MOD8CHK	LDA	A1L	
FDAF:29 07	370	AND	#\$07	
FDB1:D0 03 FDB6	371	BNE	DATAOUT	
FDB3:20 92 FD	372 XAM	JSR	PRA1	
FDB6:A9 A0	373 DATAOUT	LDA	#\$A0	
FDB8:20 ED FD	374	JSR	COUT	;OUTPUT BLANK
FDBB:B1 3C	375	LDA	(A1L),Y	
FDBD:20 DA FD	376	JSR	PRBYTE	;OUTPUT BYTE IN HEX
FDC0:20 BA FC	377	JSR	NXTA1	
FDC3:90 E8 FDDA	378	BCC	MOD8CHK	;NOT DONE YET. GO CHECK MOD 8
FDC5:60	379 RTS4C	RTS		;DONE.
FDC6:	380 *			
FDC6:4A	381 XAMPM	LSR	A	;DETERMINE IF MONITOR MODE IS
FDC7:90 EA FDB3	382	BCC	XAM	; EXAMINE, ADD OR SUBTRACT
FDC9:4A	383	LSR	A	
FDCA:4A	384	LSR	A	
FDCB:A5 3E	385	LDA	A2L	
FDCD:90 02 FDD1	386	BCC	ADD	
FDCF:49 FF	387	EOR	#\$FF	;FORM 2'S COMPLEMENT FOR SUBTRACT.
FDD1:65 3C	388 ADD	ADC	A1L	
FDD3:48	389	PHA		
FDD4:A9 BD	390	LDA	#\$BD	;PRINT '=', THEN RESULT
FDD6:20 ED FD	391	JSR	COUT	
FDD9:68	392	PLA		
FDDA:	393 *			
FDDA:48	394 PRBYTE	PHA		;PRINT BYTE AS 2 HEX DIGITS
FDDB:4A	395	LSR	A	; (DESTROYS A-REG)
FDDC:4A	396	LSR	A	
FDDD:4A	397	LSR	A	
FDDF:4A	398	LSR	A	
FDDF:20 E5 FD	399	JSR	PRHEXZ	
FDE2:68	400	PLA		
FDE3:	401 *			
FDE3:29 0F	402 PRHEX	AND	#\$0F	;PRINT HEX DIGIT IN A-REG
FDE5:09 B0	403 PRHEXZ	ORA	#\$B0	;LSBITS ONLY.
FDE7:C9 BA	404	CMF	#\$BA	
FDE9:90 02 FDED	405	BCC	COUT	
FDEB:69 06	406	ADC	#\$06	
FDED:	407 *			



FDED:6C 36 00	408	COUT	JMP	(CSWL)	;VECTOR TO USER OUTPUT ROUTINE
FDFD:	409	*			
FDF0:2C 7B 06	410	COUT1	BIT	VFACTV	;video firmware active?
FDF3:4C B4 FB	411		JMP	DOCOUT1	;mask II mode characters
FDF6:84 35	412	COUTZ	STY	YSAV1	;SAVE Y-REG
FDF8:48	413		PHA		;SAVE A-REG
FDF9:20 78 FB	414		JSR	VIDWAIT	;OUTPUT CHR AND CHECK FOR CTRL-S
FDFC:68	415		PLA		;RESTORE A-REG
FDFD:A4 35	416		LDY	YSAV1	;AND Y-REG
FDFE:60	417		RTS		;RETURN TO SENDER...
FE00:	418	*			
FE00:C6 34	419	BL1	DEC	YSAV	
FE02:F0 9F PDA3	420		BEQ	XAM8	
FE04:	421	*			
FE04:CA	422	BLANK	DEX		;BLANK TO MON
FE05:D0 16 FE1D	423		BNE	SETMDZ	;AFTER BLANK
FE07:C9 BA	424		CMF	#\$BA	;DATA STORE MODE?
FE09:D0 BB FDC6	425		BNE	XAMPM	; NO; XAM, ADD, OR SUBTRACT.
FE0B:	426	*			
FE0B:85 31	427	STOR	STA	MODE	;KEEP IN STORE MODE
FE0D:A5 3E	428		LDA	A2L	
FE0F:91 40	429		STA	(A3L),Y	;STORE AS LOW BYTE AT (A3)
FE11:E6 40	430		INC	A3L	
FE13:D0 02 FE17	431		BNE	RTS5	;INCR A3, RETURN.
FE15:E6 41	432		INC	A3H	
FE17:60	433	RTS5	RTS		
FE18:	434	*			
FE18:A4 34	435	SETMODE	LDY	YSAV	;SAVE CONVERTED ':', '+',
FE1A:B9 FF 01	436		LDA	IN-1,Y	; '- ', '.' AS MODE
FE1D:85 31	437	SETMDZ	STA	MODE	
FE1F:60	438		RTS		
FE20:	439	*			
FE20:A2 01	440	LT	LDX	#\$01	
FE22:B5 3E	441	LT2	LDA	A2L,X	;COPY A2 (2 BYTES) TO
FE24:95 42	442		STA	A4L,X	; A4 AND A5
FE26:95 44	443		STA	A5L,X	
FE28:CA	444		DEX		
FE29:10 F7 FE22	445		BPL	LT2	
FE2B:60	446		RTS		
FE2C:	447	*			
FE2C:B1 3C	448	MOVE	LDA	(A1L),Y	;MOVE (A1) THRU (A2) TO (A4)
FE2E:91 42	449		STA	(A4L),Y	
FE30:20 B4 FC	450		JSR	NXTA4	
FE33:90 F7 FE2C	451		BCC	MOVE	
FE35:60	452		RTS		
FE36:	453	*			
FE36:B1 3C	454	VERIFY	LDA	(A1L),Y	;VERIFY (A1) THRU (A2)
FE38:D1 42	455		CMF	(A4L),Y	; WITH (A4)
FE3A:F0 1C FE58	456		BEQ	VFYOK	
FE3C:20 92 FD	457		JSR	PRA1	
FE3F:B1 3C	458		LDA	(A1L),Y	
FE41:20 DA FD	459		JSR	PRBYTE	
FE44:A9 A0	460		LDA	#\$A0	
FE46:20 ED FD	461		JSR	COUT	
FE49:A9 A8	462		LDA	#\$A8	
FE4B:20 ED FD	463		JSR	COUT	
FE4E:B1 42	464		LDA	(A4L),Y	
FE50:20 DA FD	465		JSR	PRBYTE	



FE53:A9 A9	466	LDA	#\$A9	
FE55:20 ED FD	467	JSR	COUT	
FE58:20 B4 FC	468 VFYOK	JSR	NXTA4	
FE58:90 D9 FE36	469	BCC	VERIFY	
FE5D:60	470	RTS		
FE5E:	471 *			
FE5E:20 75 FE	472 LIST	JSR	AIPC	;MOVE A1 (2 BYTES) TO
FE61:A9 14	473	LDA	#\$14	; PC IF SPEC'D AND
FE63:48	474 LIST2	PHA		; DISASSEMBLE 20 INSTRUCTIONS.
FE64:20 D0 F8	475	JSR	INSTDSP	
FE67:20 53 F9	476	JSR	PCADJ	;ADJUST PC AFTER EACH INSTRUCTION.
FE6A:85 3A	477	STA	PCL	
FE6C:84 3B	478	STY	PCN	
FE6E:68	479	PLA		
FE6F:38	480	SEC		
FE70:E9 01	481	SBC	#\$01	;NEXT OF 20 INSTRUCTIONS
FE72:D0 EF FE63	482	BNE	LIST2	
FE74:60	483	RTS		
FE75:	484 *			
FE75:8A	485 AIPC	TXA		;IF USER SPECIFIED AN ADDRESS,
FE76:F0 07 FE7F	486	BEQ	AIPCRTS	; COPY IT FROM A1 TO PC.
FE78:B5 3C	487 AIPCPLP	LDA	A1L,X	;YEP, SO COPY IT.
FE7A:95 3A	488	STA	PCL,X	
FE7C:CA	489	DEX		
FE7D:10 F9 FE78	490	BPL	AIPCPLP	
FE7F:60	491 AIPCRTS	RTS		
FE80:	492 *			
FE80:A0 3F	493 SETINV	LDY	#\$3F	;SET FOR INVERSE VID
FE82:D0 02 FE86	494	BNE	SETIFLG	; VIA GOUT1
FE84:A0 FF	495 SETNORM	LDY	#\$FF	;SET FOR NORMAL VID
FE86:84 32	496 SETIFLG	STY	INVFLG	
FE88:60	497	RTS		
FE89:	498 *			
FE89:A9 00	499 SETKBD	LDA	#\$00	;DO 'IN#0'
FE8B:85 3E	500 INPORT	STA	A2L	;DO 'IN#AREG'
FE8D:A2 38	501 INPRT	LDX	#KSWL	
FE8F:A0 1B	502	LDY	#KEYIN	
FE91:D0 08 FE98	503	BNE	IOPRT	
FE93:	504 *			
FE93:A9 00	505 SETVID	LDA	#\$0	;DO 'PR#0'
FE95:85 3E	506 OUTPORT	STA	A2L	;DO 'PR#AREG'
FE97:A2 36	507 OUTPRT	LDX	#CSWL	
FE99:A0 F0	508	LDY	#COUT1	
FE9B:A5 3E	509 IOPRT	LDA	A2L	
FE9D:29 0F	510	AND	#\$0F	
FE9F:D0 06 FEA7	511	BNE	NOTPRTO	;not slot 0
FEA1:C0 1B	512	CPY	#KEYIN	;Continue if KEYIN
FEA3:F0 39 FEDE	513	BEQ	IOPRT1	
FEA5:80 1B FEC2	514	BRA	OPRTO	;=>do PR#0
FEA7:09 C0	515 NOTPRTO	ORA	#<IOADR	
FEA9:A0 00	516	LDY	#\$00	
FEAB:94 00	517 IOPRT2	STY	LOC0,X	
FEAD:95 01	518	STA	LOC1,X	
FEAF:60	519	RTS		
FEB0:	520 *			
FEB0:4C 00 E0	521 XBASIC	JMP	BASIC	;TO BASIC, COLD START
FEB3:	522 *			
FEB3:4C 03 E0	523 BASCONT	JMP	BASIC2	;TO BASIC, WARM START

FEB6:	524 *			
FEB6:20 75 FE	525 GO	JSR	ALPC	;ADDR TO PC IF SPECIFIED
FEB9:20 3F FF	526	JSR	RESTORE	;RESTORE FAKE REGISTERS
FEBC:6C 3A 00	527	JMP	(PCL)	; AND GO!
FEBF:	528 *			
FEBF:4C D7 FA	529 REGZ	JMP	REGDSP	;GO DISPLAY REGISTERS
FEC2:	530 *			
FEC2:3A	531 OPRT0	DEC	A	;Need \$FF
FEC3:8D FB 07	532	STA	CURSOR	;set checkerboard cursor
FEC6:A9 F7	533	LDA	#\$FF-M.CTL	;reset mode
FEC8:80 04 FECE	534	BRA	DOPRO	
FECA:	535 *			
FECA:4C F8 03	536 USR	JMP	USRADR	;JUMP TO CONTROL-Y VECTOR IN RAM
FECD:	537 *			
FECD:60	538 WRITE	RTS		;Tape write not needed
FECE:	539 *			
FECE:8D 7B 06	540 DOPRO	STA	VFACTV	;say video firmware inactive
FED1:8D 0E C0	541	STA	CLRALTCHAR	;switch in normal char set
FED4:0C FB 04	542	TSB	VMODE	;don't change M.CTL
FED7:DA	543	PHX		;save X and Y
FED8:5A	544	PHY		;for rest of PR#0
FED9:20 CD CD	545	JSR	CHK80	;convert to 40 if needed
FEDC:7A	546	PLY		
FEDD:FA	547	PLX		
FEDE:A9 FD	548 IOPRT1	LDA	#<OUT1	;set I/O page
FEEO:80 C9 FEAB	549	BRA	IOPRT2	;=>go set output hook
FEE2:	550 *			
FEE2:	551 *			;DECCH decrements the current cursor
FEE2:	552 *			;CLRCH sets all cursors to 0
FEE2:	553 *			;SETCUR sets cursors to value in Acc.
FEE2:	554 *			;See explanatory note with GETCUR
FEE2:	555 *			
FEE2:5A	556 DECCH	PHY		;(from \$FC10)
FEE3:20 9D CC	557	JSR	GETCUR	;get current CH
FEE6:88	558	DEY		;decrement it
FEE7:80 05 FEEF	559	BRA	SETCUR1	;go update cursors
FEE9:	560 *			
FEE9:A9 01	561 CLRCH	LDA	#1	;set all cursors to 0
FEED:3A	562 WDTNCH	DEC	A	;dec window width (from \$FC17)
FEED:5A	563 SETCUR	PHY		;save Y
FEED:AB	564	TAY		;need value in Y
FEED:20 AD CC	565 SETCUR1	JSR	GETCUR2	;save new CH
FEF1:7A	566	PLY		;restore Y
FEF2:AD 7B 05	567	LDA	OURCH	;and get new CH into acc
FEF5:60	568	RTS		;(Need LDA to set flags)
FEF6:	569 *			
FEF6:20 00 FE	570 CRMON	JSR	BL1	;HANDLE CR AS BLANK
FEF9:68	571	PLA		; THEN POP STACK
FEFA:68	572	PLA		; AND RETURN TO MON
FEFB:D0 6C FEF9	573	BNE	MONZ	;(ALWAYS)
FEFD:	574 *			
FEFD:60	575 READ	RTS		;Tape read not needed
FEFE:	576 *			
FEFE:	577 *			;OPTBL is a table containing the new opcodes that
FEFE:	578 *			;wouldn't fit into the existing lookup table.
FEFE:	579 *			
FEFE:12	580 OPTBL	DFB	\$12	;ORA (ZPAG)
FEFF:14	581	DFB	\$14	;TRB ZPAG

FF00:1A	582	DFB	\$1A	;INC A
FF01:1C	583	DFB	\$1C	;TRB ABS
FF02:32	584	DFB	\$32	;AND (ZPAG)
FF03:34	585	DFB	\$34	;BIT ZPAG,X
FF04:3A	586	DFB	\$3A	;DEC A
FF05:3C	587	DFB	\$3C	;BIT ABS,X
FF06:52	588	DFB	\$52	;EOR (ZPAG)
FF07:5A	589	DFB	\$5A	;PHY
FF08:64	590	DFB	\$64	;STZ ZPAG
FF09:72	591	DFB	\$72	;ADC (ZPAG)
FF0A:74	592	DFB	\$74	;STZ ZPAG,X
FF0B:7A	593	DFB	\$7A	;PLY
FF0C:7C	594	DFB	\$7C	;JMP (ABS,X)
FF0D:89	595	DFB	\$89	;BIT IMM
FF0E:92	596	DFB	\$92	;STA (ZPAG)
FF0F:9C	597	DFB	\$9C	;STZ ABS
FF10:9E	598	DFB	\$9E	;STZ ABS,X
FF11:B2	599	DFB	\$B2	;LDA (ZPAG)
FF12:D2	600	DFB	\$D2	;CMP (ZPAG)
FF13:F2	601	DFB	\$F2	;SBC (ZPAG)
FF14:FC	602	DFB	\$FC	;??? (the unknown opcode)
FF15:	0016 603	NUMOPS	EQU	*-OPTBL-1 ;number of bytes to check
FF15:	604	*		
FF15:	605	*		INDX contains pointers to the mnemonics for each of
FF15:	606	*		the opcodes in OPTBL. Pointers with BIT 7
FF15:	607	*		set indicate extensions to MNEML or MNEMR.
FF15:	608	*		
FF15:38	609	INDX	DFB	\$38
FF16:FB	610		DFB	\$FB
FF17:37	611		DFB	\$37
FF18:FB	612		DFB	\$FB
FF19:39	613		DFB	\$39
FF1A:21	614		DFB	\$21
FF1B:36	615		DFB	\$36
FF1C:21	616		DFB	\$21
FF1D:3A	617		DFB	\$3A
FF1E:F8	618		DFB	\$F8
FF1F:FA	619		DFB	\$FA
FF20:3B	620		DFB	\$3B
FF21:FA	621		DFB	\$FA
FF22:F9	622		DFB	\$F9
FF23:22	623		DFB	\$22
FF24:21	624		DFB	\$21
FF25:3C	625		DFB	\$3C
FF26:FA	626		DFB	\$FA
FF27:FA	627		DFB	\$FA
FF28:3D	628		DFB	\$3D
FF29:3E	629		DFB	\$3E
FF2A:3F	630		DFB	\$3F
FF2B:FC	631		DFB	\$FC ;???
FF2C:00	632		BRK	
FF2D:	633	*		
FF2D:A9 C5	634	PRERR	LDA	#\$C5 ;PRINT 'ERR', THEN FALL INTO
FF2F:20 ED FD	635		JSR	COUT ; FWEEPER.
FF32:A9 D2	636		LDA	#\$D2
FF34:20 ED FD	637		JSR	COUT
FF37:20 ED FD	638		JSR	COUT
FF3A:	639	*		

FF3A:A9 87	640 BELL	LDA	\$\$87	;MAKE A JOYFUL NOISE, THEN RETURN.
FF3C:4C ED FD	641	JMP	COUT	
FF3F:	642 *			
FF3F:A5 48	643 RESTORE	LDA	STATUS	;RESTORE 6502 REGISTER CONTENTS
FF41:48	644	PHA		; USED BY DEBUG SOFTWARE
FF42:A5 45	645	LDA	A5H	
FF44:A6 46	646 RESTRI	LDX	XREG	
FF46:A4 47	647	LDY	YREG	
FF48:28	648	PLP		
FF49:60	649	RTS		
FF4A:	650 *			
FF4A:85 45	651 SAVE	STA	A5H	;SAVE 6502 REGISTER CONTENTS
FF4C:86 46	652 SAVI	STX	XREG	; FOR DEBUG SOFTWARE
FF4E:84 47	653	STY	YREG	
FF50:08	654	PHP		
FF51:68	655	PLA		
FF52:85 48	656	STA	STATUS	
FF54:BA	657	TSX		
FF55:86 49	658	STX	SPNT	
FF57:D8	659	CLD		
FF58:60	660	RTS		
FF59:	661 *			
FF59:20 84 FE	662 OLDST	JSR	SETNORM	;SET SCREEN MODE
FF5C:20 2F FE	663	JSR	INIT	; AND INIT KBD/SCREEN
FF5F:20 93 FE	664	JSR	SETVID	; AS I/O DEVS.
FF62:20 89 FE	665	JSR	SETKBD	
FF65:	666 *			
FF65:D8	667 MON	CLD		;MUST SET HEX MODE!
FF66:20 3A FF	668	JSR	BELL	;FWEEPER.
FF69:A9 AA	669 MONZ	LDA	\$\$AA	; '*' PROMPT FOR MONITOR
FF6B:85 33	670	STA	PROMPT	
FF6D:20 67 FD	671	JSR	GETLNZ	;READ A LINE OF INPUT
FF70:20 C7 FF	672	JSR	ZMODE	;CLEAR MONITOR MODE, SCAN IDX
FF73:20 A7 FF	673 NXTIIM	JSR	GETNUM	;GET ITEM, NON-HEX
FF76:84 34	674	STY	YSAV	; CHAR IN A-REG.
FF78:A0 13	675	LDY	\$SUBTBL-CHRTBL	; X-REG=0 IF NO HEX INPUT
FF7A:88	676 CHRSRCH	DEY		
FF7B:30 E8 FF65	677	BMI	MON	;COMMAND NOT FOUND, BEEP & TRY AGAIN.
FF7D:D9 CD FF	678	CMP	CHRTBL,Y	;FIND COMMAND CHAR IN TABLE
FF80:D0 F8 FF7A	679	BNE	CHRSRCH	;NOT THIS TIME
FF82:20 BE FF	680	JSR	TOSUB	;GOT IT! CALL CORRESPONDING SUBROUTINE
FF85:A4 34	681	LDY	YSAV	;PROCESS NEXT ENTRY ON HIS LINE
FF87:4C 73 FF	682	JMP	NXTIIM	
FF8A:	683 *			
FF8A:A2 03	684 DIG	LDX	\$\$03	
FF8C:0A	685	ASL	A	
FF8D:0A	686	ASL	A	;GOT HEX DIGIT,
FF8E:0A	687	ASL	A	; SHIFT INTO A2
FF8F:0A	688	ASL	A	
FF90:0A	689 NXIBIT	ASL	A	
FF91:26 3E	690	ROL	A2L	
FF93:26 3F	691	ROL	A2H	
FF95:CA	692	DEX		;LEAVE X=\$FF IF DIG
FF96:10 F8 FF90	693	BPL	NXTBIT	
FF98:A5 31	694 NXIBAS	LDA	MODE	
FF9A:D0 06 FFA2	695	BNE	NXTBS2	;IF MODE IS ZERO,
FF9C:B5 3F	696	LDA	A2H,X	; THEN COPY A2 TO A1 AND A3
FF9E:95 3D	697	STA	A1H,X	

FFA0:95 41	698	STA	A3H,X	
FFA2:E8	699	NXTBS2	INX	
FFA3:F0 F3 FF98	700	BEQ	NXTBAS	
FFA5:D0 06 FFAD	701	BNE	NXTCHR	
FFA7:A2 00	702	GETNUM	LDX	#\$00 ;CLEAR A2
FFA9:86 3E	703	STX	A2L	
FFAB:86 3F	704	STX	A2H	
FFAD:B9 00 02	705	NXTCHR	LDA	IN,Y ;GET CHAR
FFB0:C8	706	INY		
FFB1:20 99 C3	707	JSR	UPSHIFT0	;upshift if necessary (set high bit)
FFB4:49 B0	708	EOR	#\$B0	
FFB6:C9 0A	709	CMP	#\$0A	
FFB8:90 D0 FFBA	710	BCC	DIG	;it's a digit
FFBA:80 37 FFB3	711	BRA	GETHEX	;check for other digits
FFBC:00	712	BRK		
FFBD:00	713	BRK		
FFBE:	714 *			
FFBE:A9 FE	715	TOSUB	LDA	#<CO ;DISPATCH TO SUBROUTINE, BY
FFC0:48	716	PHA		; PUSHING THE HI-ORDER SUBR ADDR,
FFC1:B9 E0 FF	717	LDA	SUBTBL,Y	; THEN THE LO-ORDER SUBR ADDR
FFC4:48	718	PHA		; ONTO THE STACK,
FFC5:A5 31	719	LDA	MODE	; (CLEARING THE MODE, SAVE THE OLD
FFC7:A0 00	720	ZMODE	LDY	#\$00 ; MODE IN A-REG),
FFC9:84 31	721	STY	MODE	
FFCB:60	722	RTS		; AND 'RTS' TO THE SUBROUTINE!
FFCC:	723 *			
FFCC:EA	724	NOP		
FFCD:	725 *			
FFCD:BC	726	CHRTBL	DFB	\$B0 ;^G (BASIC WARM START)
FFCE:B2	727	DFB	\$B2	;^Y (USER VECTOR)
FFCF:BE	728	DFB	\$BE	;^E (OPEN AND DISPLAY REGISTERS)
FFD0:EF	729	DFB	\$EF	;^V (MEMORY VERIFY)
FFD1:C4	730	DFB	\$C4	;^K (IN\$SLOT)
FFD2:A9	731	DFB	\$A9	;^P (PR\$SLOT)
FFD3:BB	732	DFB	\$BB	;^B (BASIC COLD START)
FFD4:A6	733	DFB	\$A6	;^- (SUBTRACTION)
FFD5:A4	734	DFB	\$A4	;^+ (ADDITION)
FFD6:06	735	DFB	\$06	;^M (MEMORY MOVE)
FFD7:95	736	DFB	\$95	;^< (DELIMITER FOR MOVE, VFY)
FFD8:07	737	DFB	\$07	;^N (SET NORMAL VIDEO)
FFD9:02	738	DFB	\$02	;^I (SET INVERSE VIDEO)
FFDA:05	739	DFB	\$05	;^L (DISASSEMBLE 20 INSTRS)
FFDB:00	740	DFB	\$00	;^G (EXECUTE PROGRAM)
FFDC:93	741	DFB	\$93	;^' (MEMORY FILL)
FFDD:A7	742	DFB	\$A7	;^' (ADDRESS DELIMITER)
FFDE:C6	743	DFB	\$C6	;^CR (END OF INPUT)
FFDF:99	744	DFB	\$99	;BLANK
FFE0:	745 *			
FFE0:	746 *	Table of low order monitor routine		
FFE0:	747 *	dispatch addresses.		
FFE0:	748 *			
FFE0:B2	749	SUBTBL	DFB	>BASCONT-1
FFE1:C9	750	DFB		>USR-1
FFE2:BE	751	DFB		>REG2-1
FFE3:35	752	DFB		>VERIFY-1
FFE4:8C	753	DFB		>INPRT-1
FFE5:96	754	DFB		>OUTPRT-1
FFE6:AF	755	DFB		>KBASIC-1



FFE7:17		756	DFB	>SETMODE-1	
FFE8:17		757	DFB	>SETMODE-1	
FFE9:2B		758	DFB	>MOVE-1	
FFEA:1F		759	DFB	>LT-1	
FFEB:83		760	DFB	>SETNORM-1	
FFEC:7F		761	DFB	>SETINV-1	
FFED:5D		762	DFB	>LIST-1	
FFEE:B5		763	DFB	>GO-1	
FFEF:17		764	DFB	>SETMODE-1	
FFF0:17		765	DFB	>SETMODE-1	
FFF1:F5		766	DFB	>CRMON-1	
FFF2:03		767	DFB	>BLANK-1	
FFF3:		768 *			
FFF3:69 88		769	GETHEX	ADC	#\$88
FFF5:C9 FA		770		CMP	#\$FA
FFF7:80 91	FF8A	771		BCS	DIG
FFF9:60		772		RTS	
FFFA:		773 *			
FFFA:FB 03		774		DW	NMI ;NON-MASKABLE INTERRUPT VECTOR
FFFC:62 FA		775		DW	RESET ;RESET VECTOR
FFFE:03 C8		776	IRQVECT	DW	NEWIRQ ;INTERRUPT REQUEST VECTOR



3D A1H	3C A1L	FE78 A1PCLP	FE7F A1PCRTS
FE75 A1PC	3F A2H	3E A2L	41 A3H
40 A3L	43 A4H	42 A4L	45 A5H
44 A5L	45 ACC	C8FF ACDONE	04FF ACIABUF
C988 ACIADONE	G900 ACIAINT	C908 AC ATST	FDD1 ADD
FD84 ADDINP	F8F8 ADV2	?F8F4 ADVANCE	C94B AIEATIT
C943 AINOF LSH	C94D AIPASS	C922 AIPORT2	C91C AITST2
C01E ALTCHARSET	?03F5 AMPERV	FD03 APFLE2C	FB60 APFLEI1
0438 ASTAT	C6A2 BADRD1	C6D3 BADREAD	C77C BANGER
2B BAS2H	2A BAS2L	FBC1 BASCALC	FBDO BASCLC2
FEB3 BASCONT	29 BASH	E003 BASIC2	C324 BASICENT
C79F BASICIN	C317 BASICINIT	E000 BASIC	28 BASL
FD71 BCKSPC	FAA3 BEEPFX	?F8DD BELL1	FF3A BELL
FBE4 BELL2	0215 BINH	0214 BINL	C329 BINPUT
FE00 BL1	FE04 BLANK	FCDD BLAST	4F BOOTDEV
07DB BOOTSCRN	3C BOOTTMP	?C326 BPRINT	?FA4C BREAK
03FD BRKV	?FC10 BS	04 BUTMODE	C061 BUTNO
C062 BUTN1	CFC2 C03	C307 C3COUT1	?C300 C3ENTRY
C305 C3KEYIN	FD62 CANCEL	CA76 CDONE2	CA3C CDONE
?CD7D CGO	F9BA CHAR1	F9B4 CHAR2	CDGD CHK80
FBD9 CHK8ELL	C528 CHKMOU	CB4E CNKRT	FF7A CHRSRCH
24 CH	C132 CROK	FFCD CHRTBL	CA28 CKDIG
FC9E CLEOLZ	FC46 CLEOP1	CBEE CLR0	CBFC CLR1
CBF1 CLR2	CC02 CLR3	CB87 CLR40	C000 CLR80COL
C00C CLR80VID	CBDA CLR80	C00E CLRALTCHAR	?C058 CLRANO
?C05A CLRAN1	?C05C CLRAN2	?C05E CLRAN3	FEE9 CLRCH
C1DD CLRCOL	FC9C CLREOL	FC44 CLREOP2	FC42 CLREOP
FC5D CLREOP1	CB8F CLRHAF	CD9B CLRIT	CC97 CLRKBD
FCAO CLRLIN	CC04 CLRPORT	?CFFF CLRR0M	F838 CLRS02
F83C CLRSC3	?F832 CLRS0R	F836 CLRTOP	CA7D CMDB
CA5D CMDCR	BF CMDCUR	CA79 CMDD	CA68 CMDI2
CA67 CMD1	CA67 CMDK	CA14 CMDLOOP	CA67 CMDL
C9DE CMDLIST	CA5D CMDN	CAB0 CMDP2	CA78 CMDP
CAC4 CMDQ	CAB5 CMDB	CA99 CMDS	CAC6 CMDT
CB05 CMDT2	CB17 CMDT3	C9C7 CMTABLE	CA55 CMDZ
CA25 CMDZ2	CA4D CMFOUND	C555 CMLOK	C538 CMLOOP
C577 CMNOINT	C58E CMNOVBL	C57B CMNOY	C55D CMNTO
C562 CMRGHT	C56F CMROK	CA43 CMSET	C542 CMXMOV
CFB7 C01	0738 COL	FCCA COLDSTART	30 COLOR
FCE7 COM1	FCF6 COM2	FCFC COM3	CA36 COMINIT
C9EB COMMAND	C266 COMMPORT	C263 COMOUT	C200 COMSLOT
CB28 COMTBL	C338 COPYROM	C348 COPYROM2	FDF6 COUTZ
FD0D COUT	FDFO COUT1	FEF6 CRMON	?FD8B CROUT1
FD8E CROUT	FC62 CR	FC85 CRRTS	37 CSWH
36 CSWL	CD2A CTLADR	CD58 CTLCHAR	CD54 CTLCHARO
FCA4 CTLDO	CD6F CTLDONE	CD71 CTLGO	CD80 CTLGO1
14 CTLNUM	CD91 CTLOFF	CD95 CTLON	CD15 CTLTAB
07FB CURSOR	C51D CVBUT	C516 CVM0VED	C4ED CVNOVBL
25 CV	FDB6 DATAOUT	FB8C DCX	FEE2 DECCH
C2C8 DEFAULT	C2F1 DEF0COM	C2D9 DEFFF	C2FC DEFIDX
C2CE DEFLOOP	C6D9 DENIB1	C6D7 DENIBL	C885 DEVNO
FF8A DIG	CA30 DIGLOOP	0356 DNIBL	CB82 DOCLR
FBB4 DOGOUT1	FB54 DOCTL	C6FB DODRV2	C188 DONE
FD20 DONXTCUR	FECE DOPRO	?C701 DRV2BOOT	C608 DRV2ENT
C5C3 DV10LOOP	C5CA DV10LT	C111 ENTRI	C230 ENTR
F8A1 ERR	9B ESC	CCD7 ESCO	?CCE3 ESC1
CCE5 ESC2	CCC0 ESC3	CD0C ESCCHAR	0638 ESCHAR
0013 ESCNUM	CCED ESCRDKEY	CCF8 ESCTAB	C28C EXIT1

C28A EXITX	?C65C EXTENT	C63D EXTENT1	05F9 EXTINT2
0538 EXTINT	F800 F8ORG	FBB3 F8VERSION	C142 FIXCH
?FA9B FIXSEV	06B8 FLAGS	CB1B FLUSH	F962 FMT1
F9A6 FMT2	CB67 FNDCTL	2E FORMAT	C64B FUG1
?C648 FUGIT	F847 GBASCALC	27 GBASH	26 GBASL
F856 GBCALC	C8F8 GBEMPTY	C8DB GBNOOVR	C393 GETALT1
C398 GETALT2	C37C CETALT	C8CC GETBUF	C3A6 GETCOUT
GCA7 GETCURI	CC9D GETCUR	CCAD GETCUR2	CCB7 GETCUR3
CCBF GETCURX	F8A5 GETFMT	FFF3 GETHEX	FC80 GETINDX
?FD6F GETLN1	FD67 GETLNZ	?FD6A GETLN	FFA7 GETNUM
CB57 GETST	CEFA GETX	?CF06 GETY	CF38 GKEY
?FD25 GOTKEY	FEB6 GO	C89F COBREAK	06 GOODF8
C28F GOREMOTE	C290 GOTERM	F8CC GOTONE	2C H2
C64E HANGING	C5E3 HODONE	C5BE HDLOOP	C5B8 HDPOS2
?FCC9 HEADR	C5AC HEXDEC2	C59B HEXTODEC	?C057 HIRIS
F81C HLINE1	?F819 HLINE	FC58 HOME	CDA5 HOMECUR
CE1B HOOKITUP	CE20 HOOKUP	F897 LEVEN	0200 INBUF
CAOC INCMD	FF15 INDX	?F88C INSDS2	0200 IN
C405 INENT	F82F INIT	C41C INITMOUSE	?FE8B INFORT
FE8D INFRT	F882 INSDS1	F8D0 INSTDSP	CC12 INVERT
32 INVFLG	CC1C INVX	C000 LOADR	FE9B IOPRT
FEDE IOPRT1	FEAB IOPRT2	FF58 LORTS	C058 IOU
C078 IOUDSBL	C079 IOUENBL	C806 IRQ1	C827 IRQ2
C831 IRQ3	C83B IRQ4	C850 IRQ5	C85E IRQ6
C861 IRQ7	C873 IRQ8	C88C IRQDNE1	C88F IRQDNE2
C899 IRQDNE3	C882 IRQDONE	?03FE IRQLOC	G989 IROTBLE
FFFE IRQVECT	?FA40 IRQ	C663 LSMRK1	C22F ISRBY
CFE9 JMPDEST	C32C JPINIT	C32F JPREAD	C335 JPSTAT
C332 JPWRITE	C010 KBDSTRE	FB88 KBDWAIT	C000 KBD
FD1B KEYIN	?FD18 KEYINO	39 KSWH	38 KSWL
C08B LCBANK1	C083 LCBANK2	2F LENGTH	FC66 LF
0400 LINE1	FE5E LIST	FE63 LIST2	2C LMNEM
00 LOCO	01 LOC1	FD38 LOCKPICK	C056 LORES
FE22 LT2	FE20 LT	? 40 M.40	20 M.CTL2
08 M.CTL	10 M.CURSOR	08 M.GOXY	01 M.MOUSE
80 M.PASCAL	04 M.VMODE	44 MACSTAT	C709 MAKTEL
2E MASK	C9D4 MASK1	C9D9 MASK2	05F8 MAXH
04F8 MAXL	077D MAXXH	067D MAXXL	?07FD MAXYH
?06FD MAXYL	C400 MBASIC	C79B MBBAD	0578 MINH
0478 MINL	Q57D MINXH	047D MINXL	?05FD MINYH
?04FD MINYL	C8AB MIROLP	C8C2 MIROSTD	C4F1 MISTAT
?C052 MIXCLR	C053 MIXSET	F9C0 MNEML	FA00 MNEMR
F8BE MNNDX1	F8C2 MNNDX2	F8C9 MNNDX3	FDAD MODSCHK
31 MODE	EF65 MON	EF69 MONZ	067C MOUARM
C063 MOUBUT	C048 MOUCLR	?C058 MOUDSBL	?C059 MOUENBL
07FC MOUMODE	C4D5 MOUSEINT	CD9F MOUSOFF	CD99 MOUSON
077C MOUSTAT	0478 MOUTEMP	C066 MOUX1	057C MOUXH
C015 MOUXINT	047C MOUXL	C067 MOUY1	05FC MOUYH
C017 MOUYINT	04FC MOUYL	20 MOVARM	CF86 MOVEAUX
CF99 MOVEC2M	C8A2 MOVEIRQ	CF9F MOVELOOP	FE2C MOVE
CFCB MOVERET	CF9F MOVESTRT	02 MOVMODE	C7DE MPADDLE
C72F MSG	CAA6 MSLOOP	07F8 MSL0T	CAA4 MSWAIT
0300 NBUF1	FB60 NEWADV1	FBA0 NEWADV	FA47 NEWBRK
FC99 NEWC1	FC90 NEWCLEGLZ	FC8D NEWCLREOL	FC73 NEWCR
CCCC NEWESC	C803 NEWIRQ	?FA81 NEWMON	FC38 NEWOP1
FC35 NEWOPS	FC86 NEWVTAB	FC88 NEWVTABZ	CFA9 NEXTA1
03FB NMI	CA09 NOCMD	C46B NOERROR	?FD45 NOESC1
FD4A NOESC2	FD44 NOESCAPE	C26B NOESC	PACF NOFIX

C725 NOPATRN	C371 NOREAD	CA93 NOSHIFT	C4F9 NOSTAT2
C36A NOT1	C8FE NOTACIA	FD5F NOTCR1	FD4D NOTCR
?CC68 NOTINV1	CC6B NOTINV2	CC53 NOTINV	FEA7 NOTPRT0
C22E NOTRDY	FB94 NOWAIT	047F NUMBER	0016 NUMOPS
FCBA NXTA1	FCB4 NXTA4	FF98 NXTBAS	FF90 NXTBIT
FFA2 NXTBS2	FD75 NXTCHAR	FFAD NXTCHR	?F85F NXTCOL
077B NXTCUR	FF73 NXTITM	FA59 OLDBRK	047B OLDCH
067A OLDCUR2	0679 OLDCUR	?FF59 OLDRST	FEC2 OPRT0
FEFE OPTBL	057B OURCH	05FB OURCV	C407 OUTENT
?FE95 OUTPORT	FE97 OUTPRT	C1E4 P1INIT	C1F3 P1READ2
C1EE P1READ	C1FB P1STATUS	C1F6 P1WRITE	C211 P2INIT
C213 P2READ	C217 P2STATUS	C215 P2WRITE	C064 PADDL0
CF71 PASCALC	?CF7F PASCCLC2	CC0B PASINVERT	CF35 PASREAD
C850 PASSKIP1	C97C PBFULL	C973 PBOK	?F954 PCADJ2
F95C PCADJ4	F953 PCADJ	F956 PCADJ3	3B PCH
3A PCL	CF19 PCTL	C7F6 PDOK	C7EB PDON
CC3D PICK1	CC33 PICK2	CC3F PICK3	CC4A PICK4
95 PICK	CC1D PICKY	CF41 PINIT	CEBC PIORDY
F800 PLOT	F80E PLOT1	CEC0 PNOTRDY	C402 PNULL
FD92 PRA1	F910 PRADR1	F914 PRADR2	F926 PRADR3
F92A PRADR4	F930 PRADR5	F94A PRBL2	?F94C PRBL3
F948 PRBLNK	FDDA PRBYTE	?FB1E PREAD	FB25 PREAD2
?FF2D PRERR	CEF7 PRET	?FDE3 PRHEX	FDE5 PRHEX2
F8F5 PRMN1	F8F9 PRMN2	C168 PRNOW	?F941 PRNTAX
F8DB PRNTBL	F8D4 PRNTOP	F940 PRNTYX	C14C PRNT
?F944 PRNTX	33 PROMPT	FD96 PRYX2	CF66 PS1
CF51 PSETUP	CF54 PSETUP2	CF30 PSETX	C222 PSTAT2
CEB1 PSTATUS	CEBE PSTERR	?C070 PTRIG	C967 PUTBUF
C7DA PUTINBUF	CE3B PVMODE	0488 PWDTH	CEDD PWR1
FAFD PWRCON	03F4 PWREDUP	CEF4 PWRET	CEC2 PWRITE
CEFI PWRITERET	FB12 PWRUP2	FAA6 PWRUP	C506 QLOOP
C5E8 QTBL	CE45 QUIT	CE44 QX	?C060 RD40SW
C018 RD80COL	C01F RD80VID	C63F RDADR	C016 RDALTZP
C6A8 RDAT0	C6AA RDAT1	C6BA RDAT2	C6BC RDAT3
C6CB RDAT4	C6A6 RDATA	C003 RDCARDRAM	?FD35 RDCHAR
C642 RDDHDR	C656 RDHDO	C65E RDHD1	C667 RDHD2
C671 RDHD3	?C01D RDHIRES	FDOC RDKEY	C011 RDLCBNK2
C012 RDLGRAM	C002 RDMAINRAM	?C01B RDMIX	C01C RDPAGE2
C013 RDRAMRD	C014 RDRAMWRT	C685 RDSEC1	C687 RDSEC2
C68F RDSEC3	C683 RDSECT	FAE4 RDSP1	C01A RDTEXT
?C019 RDVBLBAR	?FEFD READ	FAD7 REGDSP	FEBF REGZ
?F938 RELADR	FA62 RESET	FAED RESET.X	C354 RESETLG
FF3F RESTORE	?FF44 RESTR1	C641 RETRY1	C657 RETRY
FADA RGDSP1	FB02 RGDSP2	2D RMNEM	4F RNDH
4E RNDL	C081 ROMIN	C37B ROMOK	0478 ROMSTATE
FAD2 RTBL	F80C RTMASK	F87F RTMSKZ	F831 RTS1
FBEF RTS2B	F961 RTS2	FB2E RTS2D	FBFC RTS3
FCC8 RTS4B	?FDC3 RTS4C	?FC34 RTS4	FE17 RTS5
?FCB3 RTS6	?FF4C SAV1	FF4A SAVE	BFFB SCNTL
BFFA SCOMD	CE58 SCR1	CE5E SCR2	CE66 SCR3
CE79 SCR4	CE82 SCR5	CE8B SCR6	CE96 SCR7
?CE8D SCR8	CEAD SCR9	CEB9 SCRL3	CE9B SCRLEVEN
CBA2 SCRLFT	CE6D SCRLIN	CEB0 SCRLODD	?F871 SCRN
F879 SCRN2	CE80 SCRN48	CE53 SCRN84	CB3D SCROLLDN
CB38 SCROLLIT	CB35 SCROLLUP	?FC70 SCROLL	BFF8 SDATA
C61F SEEKZERO	C296 SERIN	C11E SERISOUT	0388 SERMODE
C18C SEROUT	C191 SEROUT2	C19D SEROUT3	C117 SERPORT
C100 SERSLOT	C146 SERVID	CDC0 SET40	C001 SET80COL

CDBE SET80	C00D SET80VID	C00F SETALTCHAR	C009 SETALTZP
?C059 SETANO	?C05B SETAN1	?C05D SETAN2	C05F SETAN3
C184 SETCH	?F864 SETCOL	F8EE SETCUR1	F8EC SETCUR
CB67 SETDBAS	?FB40 SETGR	CE23 SETHOOKS	F886 SETIFLG
FE80 SETINV	?C454 SETIOU	CDA1 SETIT	F889 SETKBD
FE1D SETMDZ	FE18 SETMODE	FE84 SETNORM	?FAA9 SETPG3
FAAB SETPLP	?FB6F SETPWRC	C360 SEIROM	CB88 SETSRC
C008 SETSTDZP	CACD SEITERM	?FB39 SETTXT	C233 SETUP
CB83 SETUP2	FE93 SETVID	FB4B SETWND	CE1A SETX
CB01 SEVI	C44C SHOWCUR	C45E SILOOP	C28E SINOMOD
C205 SIN	C465 SINOCH	CBA8 SKPLFT	CB84 SKPRT
2B SLO TZ	C1 SLTDMY	C46C SMINVALID	C18B SOGMD
C1E2 SODONE	03F2 SOFTEV	C1BF SOMAIN	C1A9 SORDY
C1C1 SORDY2	C207 SOUT	C030 SPKR	49 SPNT
BFF9 SSTAT	CF29 STARTXY	48 STATUS	CAE6 STCLR
FB65 STITLE	?FE0B STOR	FBF0 STORADV	C388 STORCH
C3DB STORE1	?C3F7 STORE4	C3C1 STORE	C3EE STORE2
?C3F2 STORE3	C3F9 STORE5	C3B3 STORY	CB21 STRTS
CAED STSET	CAF6 STWASOK	FFB0 SUBTBL	C246 SUDODEF
C25C SUDONE	C249 SUMODEF	C257 SUOUT	C160 TAB
?FB5B TABV	?C020 TAPEOUT	C740 TBL1	C749 TBL2
C71B TBLLOOP2	C70D TBLLOOP	04F8 TEMP1	06F8 TEMP
0578 TEMPA	05F8 TEMPY	C293 TERM1	DF TERMCUR
C275 TESTKBD	0800 THBUF	?FB09 TITLE	C15E TOOFAR
FFBE TOSUB	06FF TRKEY	067F TRSER	05FF TWKEY
057F TWSER	C050 TXTCLR	C054 TXTPAGE1	C055 TXTPAGE2
C051 TXTSET	05FA TYPHED	CC91 UD2	C39B UPSHIFT
FG1A UP	CC70 UPDATE	C399 UPSHIFT0	03F8 USRADR
FECA USR	2D V2	C070 VBLCLR	C019 VBLINT
OC VBLMODE	FE36 VERIFY	067B VFACIV	FE58 VFYOK
CE31 VIDMODE	FC04 VIDOUT1	FBFD VIDOUT	FB78 VIDWAIT
F826 VLINEZ	F82B VLINE	04FB VMODE	FC30 VTAB40
FC22 VTAB	FB59 VTAB23	FC24 VTABZ	FCA8 WAIT
FCA9 WAIT2	FCAA WAIT3	FEEB WDTCH	CDD5 WIN0
CDE0 WIN1	CDED WIN2	CDF2 WIN3	C202 WIN4
CDD2 WIN40	CE18 WIN5	CDD4 WIN80	23 WNDBTM
20 WNDLFT	CE0A WNDREST	22 WNDTOP	21 WNDWDTH
C005 WRCARDRAM	?FECB WRITE	C004 WRMAINRAM	CD8D X.CUR.OFF
CD89 X.CUR.ON	CDB7 X.SI	CDB0 X.SO	C3A5 X.UPSHIFT
FD83 XAM	FDA3 XAM8	FDC6 XAMPM	FEBO XBASIC
C9AD XBITKBD	C9CD XKBK1	C9C2 XKBK2	06FB XCOORD
CFE6 XFERA2P	CFE0 XFERC2M	CFE6 XFERZP	CPCD XFER
C752 XLOOP1	C780 XMBASIC	C78D XMBOUT	C490 XMCBONE
C4B0 XMCLAMP	C484 XMCLEAR	C59A XMDONE	C473 XMHZ
C471 XMHLOOP	C46D XMHOME	C4CF XMINT	C763 XMODE
C495 XMREAD	C50C XMSKIP	C4FC XMTSTINT	C9A8 XNOKEY
C8F0 XNOSBUF	91 XON	C758 XPAGE	C4A4 XRBUT
C4AB XRBUT2	C98F XRDKBD	C8C5 XRDSE	C8FF XRDSNO
46 XREG	C9A0 XRKBD1	C423 XRLOOP	C766 XRSET
C76E XRSI1	C43D XSETMOU	C452 XSOFF	?C100 XXX
0008 YHI	47 YREG	35 YSAV1	34 YSAV
FFC7 ZMODE	C7FB ZZNM1	CB24 ZZNM2	CB4D ZZQUIT

\*\* SUCCESSFUL ASSEMBLY := NO ERRORS  
 \*\* ASSEMBLER CREATED ON 15-JAN-84 21:28  
 \*\* TOTAL LINES ASSEMBLED 4406  
 \*\* FREE SPACE PAGE COUNT 47





# *Glossary*



**65C02:** The microprocessor used in the Apple IIc computer.

**ACIA:** Asynchronous Communications Interface Adapter. A single chip that converts data from parallel to serial form, and vice versa, and handles serial transmission and reception and RS-232-C signals, under the control of its internal registers set and changed by firmware or software.

**accumulator:** The register in the 6502 and 65C02 microprocessors where most computations are performed.

**acronym:** A word formed from the initial letters of a name or phrase, such as ROM, from read-only memory.

**ADC:** See **analog-to-digital converter**.

**address:** A number used to identify something, such as a location in the computer's memory.

**analog:** Represented in terms of a physical quantity that can vary smoothly and continuously over a range of values. For example, a conventional 12-hour clock face is an analog device that represents the time of day in terms of the angles of the clock's hands. Compare **digital**.

**analog-to-digital converter:** A device that converts quantities from analog to digital form. For example, the Apple IIc's hand control converts the position of the control dial (an analog quantity) into a discrete number (a digital quantity) that changes in steps even when the dial is turned smoothly.

**AND:** A logical operator that produces a true result if both of its operands are true, a false result if either or both of its operands are false; compare **OR**, **exclusive OR**, **NOT**.

**Apple IIc:** A personal computer in the Apple II family, manufactured and sold by Apple Computer, Inc.

**Applesoft:** An extended version of the BASIC programming language used with the Apple IIc computer. The firmware for interpreting and executing programs in Applesoft is included in the Apple IIc ROM.

**ASCII:** American Standard Code for Information Interchange; a code in which the numbers from 0 to 127 stand for text characters, used for representing text inside a computer and for transmitting text between computers or between a computer and a peripheral device.

**assembler:** A language translator that converts a program written in assembly language into an equivalent program in machine language.

**assembly language:** A low-level programming language in which individual machine-language instructions are written in a symbolic form more easily understood by a human programmer than machine language itself.

**asserted:** Made true (positive in positive-true logic; negative in negative-true logic).

**asynchronous:** Having a variable time interval between characters.

**back panel:** The rear face of the Apple IIc computer, which includes the power switch, the power connector, and connectors for two serial devices, a video display device, an external disk drive, and a mouse or hand control.

**bandwidth:** A measure of the range of frequencies a device can handle. In the case of a video monitor, greater bandwidth enables it to display more information; to display 80 columns of text, a monitor should have a bandwidth of at least 12 MHz.

**base address:** In indexed addressing, the fixed component of an address.

**baud:** A unit of signaling speed equal to the number of discrete conditions or signal events per second. Often equated (though not precisely) with bits per second.

**binary:** The representation of numbers in terms of powers of two, using the two digits 0 and 1. Commonly used in computers, since the values 0 and 1 can easily be represented in physical form in a variety of ways, such as the presence or absence of current, positive or negative voltage, or a white or black dot on the display screen.

**bit:** A binary digit (0 or 1); the smallest possible unit of information, consisting of a simple two-way choice, such as yes or no, on or off, positive or negative, something or nothing.

**board:** See **printed-circuit board**.

**boot:** To start up a computer by loading a program into memory from an external storage medium such as a disk. Often accomplished by first loading a small program whose purpose is to read the larger program into memory. The program is said to *pull itself up by its own bootstraps*.

**bootstrap:** See **boot**.

**BREAK:** A SPACE (0) signal sent over a communication line, of long enough duration to interrupt the sender. This signal is often used to end a session with a timesharing service.

**BRK:** A 65C02 instruction that causes the microprocessor to halt.

**buffer:** An area of the computer's memory used as a *holding area* where information can be stored by one program or device and then read out by another at a different speed.

**bus:** A group of wires that transmit related information from one part of a computer system to another. In the Apple IIc, the address bus has 16 wires, and the data bus has eight.

**byte:** A unit of information consisting of a fixed number of bits; on the Apple IIc, one byte consists of eight bits and can represent any value between 0 and 255.

**carriage return:** An ASCII character (decimal 13; Appendix H) that ordinarily causes a printer or display device to place the subsequent character on the left margin. On a manual typewriter, this movement is combined with line feed (the advancement of the paper to the next line). With computers, carriage return and line feed are separate, causing hair-raising problems for the user.

**carrier:** The background signal on a communication channel that is modified to *carry* the information. Under RS-232-C rules, the carrier signal is equivalent to a continuous MARK (1) signal; a transition to 0 then represents a start bit.

**carry flag:** The C bit in the 65C02 processor status register, used to hold the *carry bit* in addition and subtraction.

**cathode-ray tube:** An electronic device, such as a television picture tube, that produces images on a screen coated with phosphors that emit light when struck by a focused beam of electrons.

**central processing unit:** See **processor**.

**character:** A letter, digit, punctuation mark, or other symbol used in printing, displaying or transferring information.

**character code:** A number used to represent a text character for processing by a computer system.

**chip:** The small piece of semiconducting material (usually silicon) on which an integrated circuit is fabricated.

**Clear To Send:** An RS-232-C signal from a DCE to a DTE that is normally kept false until the DCE makes it true, indicating that all circuits are ready to transfer data out.

**code:** (1) A number or symbol used to represent some piece of information in a compact or easily processed form. (2) The statements or instructions making up a program.

**cold start:** The process of starting up the Apple IIc when the power is first turned on (or as if the power had just been turned on) by loading the operating system into main memory, then loading and running a program. Compare **warm start**.

**command:** A communication from the user to a computer system (usually typed from the keyboard) directing it to perform some action.

**command character:** An ASCII character, usually **CONTROL-A** or **CONTROL-I**, that causes the serial port firmware to interpret subsequent characters as a command.

**command register:** An ACIA location (at address \$C09A for port 1 and \$C0AA for port 2) that stores parity type and RS-232-C signal characteristics.

**communication mode:** An operating state in which serial port 2 (or 1, if so set) is prepared to exchange data and signals with a DCE (such as a modem).

**compiler:** A language translator that converts a program written in a high-level programming language into an equivalent program in some lower-level language (such as machine language) for later execution. Compare **interpreter**.



**composite video:** A video signal that includes both display information and the synchronization (and other) signals needed to display it.

**computer:** An electronic device for performing predefined (programmed) computations at high speed and with great accuracy.

**computer system:** A computer and its associated hardware, firmware, and software.

**connector:** A physical device such as a plug, socket, or jack, used to connect two devices to one another.

**control character:** A character that controls or modifies the way information is printed or displayed. Control characters have ASCII codes between \$00 and \$1F (or between \$80 and \$9F if the high-order bit is set). You can generate them at the Apple IIc keyboard by holding down **CONTROL** while typing one of the letter keys or @ [ \ ] ^ or \_.

**control register:** An ACIA location (at address \$C09B for port 1, or \$C0AB for port 2) that stores data format and baud rate selections.

**CPU:** Central processing unit; see **processor**.

**CRT:** See **cathode-ray tube**.

**cursor:** A symbol displayed on the screen that marks where the user's next action will take effect or where the next character typed from the keyboard will appear.

**DAC:** See **digital-to-analog converter**.

**data:** Information; especially information used or operated on by a program.

**data bit:** One of five to eight bits representing a character.

**Data Carrier Detect:** An RS-232-C signal from a DCE (such as a modem) to a DTE (such as an Apple IIc) indicating that a communication connection has been established.

**Data Communication Equipment:** As defined by the RS-232-C standard, any device that transmits or receives information. Usually this is a modem. However, when a Modem Eliminator is used, the Apple IIc itself looks like a DCE to the other device, and the other device looks like a DCE to the Apple IIc.

**data format:** The form in which data is stored, manipulated or transferred. Serial data transmitted and received by port 1 or 2 has a data format of: one start bit, five to eight data bits, an optional parity bit, and one, one and a half, or two stop bits.

**Data Set Ready:** An RS-232-C signal from a DCE to a DTE indicating that the DCE has established a connection.

**Data Terminal Equipment:** As defined by the RS-232-C standard, any device that generates or absorbs information, thus acting as a terminus of a communication connection.

**Data Terminal Ready:** An RS-232-C signal from a DTE to a DCE indicating a readiness to transmit or receive data.

**DCD:** See **Data Carrier Detect**.

**DCE:** See **Data Communication Equipment**.

**debug:** To locate and correct an error or the cause of a problem or malfunction in a computer system. Typically used to refer to software-related problems.

**decimal:** The common form of number representation used in everyday life, in which numbers are expressed in terms of powers of ten, using the ten digits 0 to 9.

**default:** A value, action, or setting that is assumed or set in the absence of explicit instructions otherwise.

**demodulate:** To recover the information being transmitted by a modulated signal; for example, a conventional radio receiver demodulates an incoming broadcast signal to convert it into sound emitted by a speaker.

**device:** (1) A physical apparatus for performing a particular task or achieving a particular purpose. (2) In particular, a hardware component of a computer system.

**digit:** (1) One of the characters 0 to 9, used to express numbers in decimal form. (2) One of the characters used to express numbers in some other form, such as 0 and 1 in binary or 0 to 9 and A to F in hexadecimal.

**digital:** Represented in a discrete (noncontinuous) form, such as numerical digits. For example, contemporary digital clocks display the time in numerical form (such as 2:57) instead of using the positions of a pair of hands on a clock face. Compare **analog**.



**digital-to-analog converter:** A device that converts quantities from digital to analog form.

**DIP:** See **dual in-line package**.

**disassembler:** A language translator that converts a machine-language program into an equivalent program in assembly language, more easily understood by a human programmer. The opposite of an assembler.

**disk:** An information storage medium consisting of a flat, circular magnetic surface on which information can be recorded in the form of small magnetized spots, similarly to the way sounds are recorded on tape.

**disk drive:** A device that writes and reads information on the surface of a magnetic disk.

**diskette:** A term sometimes used for the small (5-1/4-inch) flexible disks used with the Apple Disk II drive.

**Disk II drive:** A model of disk drive made and sold by Apple Computer for use with the Apple IIe computer; uses 5-1/4-inch flexible (*floppy*) disks.

**Disk Operating System:** An optional software system for the Apple IIe that enables the computer to control and communicate with one or more Disk II drives.

**display:** (1) Information exhibited visually, especially on the screen of a display device. (2) To exhibit information visually. (3) A display device.

**display device:** A device that exhibits information visually, such as a television receiver or video monitor.

**display screen:** The glass or plastic panel on the front of a display device, on which images are displayed.

**DOS:** See **Disk Operating System**.

**DSR:** See **Data Set Ready**.

**DTE:** See **Data Terminal Equipment**.

**DTR:** See **Data Terminal Ready**.

**dual in-line package:** An integrated circuit packaged in a narrow rectangular box with a row of metal pins along each side; similar in appearance to an armored centipede.

**echo:** To send an input character to a video display, printer, or other output device.

**edit:** To change or modify; for example, to insert, remove, replace, or move text in a document.

**editor:** A program that enables the user to create and edit information of a particular form; for example, a *text editor* or a *graphics editor*.

**effective address:** In machine-language programming, the address of the memory location on which a particular instruction actually operates, which may be arrived at by indexed addressing or some other addressing method.

**emulation mode:** A manner of operating in which one computer or interface imitates another.

**even parity:** Use of an extra bit set to 0 or 1 as necessary to make the total number of 1 bits (among the data bits plus the parity bit) an even number.

**error message:** A message displayed or printed to notify the user of an error or problem in the execution of a program.

**escape mode:** A state of the Apple IIe computer, entered by pressing the **(ESC)** key, in which certain keys on the keyboard take on special meanings for positioning the cursor and controlling the display of text on the screen.

**escape sequence:** A sequence of keystrokes, beginning with **(ESC)**, used for positioning the cursor and controlling the display of text on the screen.

**exclusive OR:** A logical operator that produces a true result if one of its operands is true and the other false, a false result if its operands are both true or both false; compare **OR**, **AND**, **NOT**.

**execute:** To perform or carry out a specified action or sequence of actions, such as those described by a program.

**firmware:** Software stored permanently in hardware: programs in read-only memory (ROM). Such programs (for example, the Applesoft interpreter and the Apple IIc Monitor program) are built into the computer at the factory; they can be executed at any time but cannot be modified or erased from main memory. Compare **hardware**, **software**.

**fixed-point:** A method of representing numbers inside the computer in which the decimal point (more correctly, the binary point) is considered to occur at a fixed position within the

number. Typically, the point is considered to lie at the right end of the number, so that the number is interpreted as an integer. Compare **floating-point**.

**flexible disk:** A disk made of flexible plastic; often called a *floppy* disk. Compare **rigid disk**.

**floating-point:** A method of representing numbers inside the computer in which the decimal point (more correctly, the binary point) is permitted to *float* to different positions within the number. Some of the bits within the number itself are used to keep track of the point's position. Compare **fixed-point**.

**form feed:** An ASCII character (decimal 12; Appendix H) that causes a printer or other paper-handling device to advance to the top of the next page.

**framing error:** In serial data transfer, absence of the expected stop bit(s) at the end of a received character. The serial port 1 and 2 ACIAs record this error by setting bit 1 (FRM) of its status register to 1. The ACIA checks and records each framing error separately: if the next character is OK, the FRM bit is cleared.

**full duplex:** Capable of simultaneous two-way communication.

**graphics:** (1) Information presented in the form of pictures or images. (2) The display of pictures or images on a computer's display screen. Compare **text**.

**half duplex:** Capable of communication in one direction at a time.

**hand control:** An optional peripheral device that can be connected to the Apple IIc's hand control connector and has a rotating dial and a pushbutton; typically used to control game-playing programs, but can be used in more serious applications as well.

**hand control connector:** A 9-pin connector on the Apple IIc's back panel, used for connecting hand controls to the computer.

**hardware:** Those components of a computer system consisting of physical (electronic or mechanical) devices. Compare **software**, **firmware**.

**hertz:** The unit of frequency of vibration or oscillation, also called cycles per second; named for the physicist Heinrich Hertz and abbreviated Hz. The Apple IIc's 65C02 microprocessor operates at a clock frequency of 1 million hertz, or 1 megahertz (MHz).

**hexadecimal:** The representation of numbers in terms of powers of sixteen, using the sixteen digits 0 to 9 and A to F. Hexadecimal numbers are easier for humans to read and understand than binary numbers, but can be converted easily and directly to binary form: each hexadecimal digit corresponds to a sequence of four binary digits, or bits.

**high-level language:** A programming language that is relatively easy for humans to understand. A single statement in a high-level language typically corresponds to several instructions of machine language.

**high-order byte:** The more significant half of a memory address or other two-byte quantity. In the Apple IIc's 65C02 microprocessor, the low-order byte of an address is usually stored first and the high-order byte second.

**high-resolution graphics:** The display of graphics on the Apple IIc's display screen as a six-color array of points, 280 columns wide and 192 rows high.

**hold time:** In computer circuits, the amount of time a signal must remain valid after some related signal has been turned off; compare **setup time**.

**Hz:** See **hertz**.

**IC:** See **integrated circuit**.

**index:** (1) A number used to identify a member of a list or table by its sequential position. (2) A list or table whose entries are identified by sequential position. (3) In machine-language programming, the variable component of an indexed address, contained in an index register and added to the base address to form the effective address.

**indexed addressing:** A method of specifying memory addresses used in machine-language programming.

**index register:** A register in a computer processor that holds an index for use in indexed addressing. The Apple IIc's 65C02 microprocessor has two index registers, called the X register and the Y register.



**input:** (1) Information transferred into a computer from some external source, such as the keyboard, a disk drive, or a modem. (2) The act or process of transferring such information.

**instruction:** A unit of a machine-language or assembly-language program corresponding to a single action for the computer's processor to perform.

**integer:** A whole number, with no fractional part; represented inside the computer in fixed-point form.

**integrated circuit:** An electronic component consisting of many circuit elements fabricated on a single piece of semiconducting material, such as silicon; see **chip**.

**interface:** The devices, rules, or conventions by which one component of a system communicates with another.

**interpreter:** A language translator that reads a program written in a particular programming language and immediately carries out the actions that the program describes. Compare **compiler**.

**interrupt:** A temporary suspension in the execution of a program by a computer in order to perform some other task, typically in response to a signal from a peripheral device or other source external to the computer.

**inverse video:** The display of text on the computer's display screen in the form of black dots on a white (or other single phosphor color) background, instead of the usual white dots on a black background.

**I/O:** Input/output; the transfer of information into and out of a computer. See **input**, **output**.

**I/O device:** Input/output device; a device that transfers information into or out of a computer. See **input**, **output**, **peripheral device**.

**I/O link:** A fixed location that contains the address of an input/output subroutine in the Apple IIc Monitor program.

**K:** Two to the tenth power, or 1024 (from the Greek root *kilo*, meaning one thousand); for example, 64K equals 64 times 1024, or 65,536.

**keyboard:** The set of keys built into the Apple IIc computer, similar to a typewriter keyboard, for typing information to the computer.

**keystroke:** The act of pressing a single key or a combination of keys (such as **CONTROL**-**C**) on the Apple IIc keyboard.

**kilobyte:** A unit of information consisting of 1K (1024) bytes, or 8K (8192) bits; see **K**.

**KSW:** The symbolic name of the location in the Apple IIc's memory where the standard input link is stored; stands for *keyboard switch*. See **I/O link**.

**language:** See **programming language**.

**language translator:** A system program that reads a program written in a particular programming language and either executes it directly or converts it into some other language (such as machine language) for later execution. See **interpreter**, **compiler**, **assembler**.

**least significant bit:** The right-hand bit of a binary number as written down; its positional value is 0 or 1.

**line feed:** An ASCII character (decimal 10; Appendix H) that ordinarily causes a printer or video display to advance to the next line.

**load:** To transfer information from a peripheral storage medium (such as a disk) into main memory for use; for example, to transfer a program into memory for execution.

**local:** Nearby; capable of direct connection using wires only.

**location:** See **memory location**.

**logical operator:** An operator, such as **AND**, that combines logical values to produce a logical result.

**low-level language:** A programming language that is relatively close to the form that the computer's processor can execute directly. Low-level languages available for the Apple IIc include 65C02 machine language and 65C02 assembly language.

**low-order byte:** The less significant half of a memory address or other two-byte quantity. In the Apple IIc's 65C02 microprocessor, the low-order byte of an address is usually stored first and the high-order byte second.

**low-power Schottky:** A type of TTL integrated circuit having lower power and higher speed than a conventional TTL integrated circuit.



**low-resolution graphics:** The display of graphics on the Apple IIc's display screen as a sixteen-color array of blocks, 40 columns wide and 48 rows high.

**machine language:** The form in which instructions to a computer are stored in memory for direct execution by the computer's processor. Each model of computer processor (such as the 65C02 microprocessor used in the Apple IIc) has its own form of machine language.

**main memory:** The memory component of a computer system that is built into the computer itself and whose contents are directly accessible to the processor.

**MARK parity:** A bit of value 1 appended to a binary number for transmission. The receiving device can then check for errors by looking for this value on each character.

**memory:** A hardware component of a computer system that can store information for later retrieval; see **main memory**, **random-access memory**, **read-only memory**, **read-write memory**.

**memory location:** A unit of main memory that is identified by an address and can hold a single item of information of a fixed size; in the Apple IIc, a memory location holds one byte, or eight bits, of information.

**MHz:** Megahertz; one million hertz. See **hertz**.

**microcomputer:** A computer, such as the Apple IIc, whose processor is a microprocessor.

**microprocessor:** A computer processor contained in a single integrated circuit, such as the 65C02 microprocessor used in the Apple IIc.

**microsecond:** One millionth of a second; abbreviated **us**.

**millisecond:** One thousandth of a second; abbreviated **ms**.

**mode:** A state of a computer or system that determines its behavior.

**modem:** Modulator/demodulator; a peripheral device that enables the computer to transmit and receive information over a telephone line; a DCE that connects a DTE to communication lines.

**modem eliminator:** The physical crossing of wires that replaces a pair of modems for direct connection of two DTEs.

**modulate:** To modify or alter a signal so as to transmit information; for example, conventional broadcast radio transmits sound by modulating the amplitude (amplitude modulation, or AM) or the frequency (frequency modulation, or FM) of a carrier signal.

**monitor:** See **video monitor**.

**Monitor program:** A system program built into the Apple IIc in firmware, used for directly inspecting or changing the contents of main memory and for operating the computer at the machine-language level.

**most significant bit:** The leftmost bit of a binary number as written down. This bit represents 0 or 1 times 2 to the power one less than the total number of bits in the binary number. For example, in the binary number 10000, which contains five digits, the 1 represents 1 times two to the fourth power—or sixteen.

**nanosecond:** One billionth (in British usage, one thousand-millionth) of a second; abbreviated ns.

**network:** A collection of interconnected, individually controlled computers, together with the hardware and software used to connect them.

**nibble:** A unit of information equal to half a byte, or four bits; can hold any value from 0 to 15. Sometimes spelled *nybble*.

**NOT:** A unary logical operator that produces a true result if its operand is false, a false result if its operand is true; compare **AND**, **OR**, **exclusive OR**.

**NTSC:** (1) National Television Standards Committee; the committee that defined the standard format used for transmitting broadcast video signals in the United States. (2) The standard video format defined by the NTSC.

**object code:** See **object program**.

**object program:** The translated form of a program produced by a language translator such as a compiler or assembler; also called object code. Compare **source program**.

**odd parity:** Use of an extra bit set to 0 or 1 as necessary to make the total number of 1 bits an odd number.

**opcode:** See **operation code**.

**operand:** A value to which an operator is applied; the value on which an opcode operates.

**operating system:** A software system that organizes the computer's resources and capabilities and makes them available to the user or to application programs running on the computer.

**operation code:** The part of a machine-language instruction that specifies the operation to be performed; often called **opcode**.

**operator:** A symbol or sequence of characters, such as  $+$  or *AND*, specifying an operation to be performed on one or more values (the operands) to produce a result.

**OR:** A logical operator that produces a true result if either or both of its operands are true, a false result if both of its operands are false; compare **exclusive OR, AND, NOT**.

**output:** Information transferred from a computer to some external destination, such as the display screen, a disk drive, a printer, or a modem.

**overflow:** A condition that occurs when the Apple IIc processor does not retrieve a received character from the ACIA's receive data register before the subsequent character arrives. The ACIA automatically sets bit 2 (OVR) of its status register; subsequent characters are lost. The receive data register contains the last valid data word received.

**page:** (1) A screenful of information on a video display, consisting on the Apple IIc of 24 lines of 40 or 80 characters each. (2) An area of main memory containing text or graphical information being displayed on the screen. (3) A segment of main memory 256 bytes long and beginning at an address that is an even multiple of 256 bytes.

**page zero:** See **zero page**.

**parallel interface:** An interface in which many bits of information (typically eight bits, or one byte) are transmitted simultaneously over different wires or channels. Compare **serial interface**.

**parity:** Maintenance of a sameness of level or count, usually the count of 1 bits in each character, for error checking.

**parity error:** Absence of the correct parity bit value in a received character. The serial port ACIAs record this error by setting bit 0 (PAR) of their status registers to 1.

**PC board:** See **printed-circuit board**.

**phase:** (1) A stage in a periodic process; a point in a cycle; for example, the 65C02 microprocessor uses a clock cycle consisting of two phases called PHI0 and PHI1. (2) The relationship between two periodic signals or processes; for example, in NTSC color video, the color of a point on the screen is expressed by the instantaneous phase of the video signal relative to the color reference signal.

**pipelining:** A feature of a processor that enables it to begin fetching the next instruction before it has finished executing the current instruction. All other things equal, processors that have this feature run faster than those without it.

**pointer:** An item of information consisting of the memory address of some other item.

**pop:** To remove the top entry from a stack.

**port:** The point of connection, usually a physical connector, between a computer and a peripheral device, another computer, or a network.

**power supply:** The hardware component of a computer that draws electrical power from a power outlet and converts it to the forms needed by some other hardware component.

**printed-circuit board:** A hardware component of a computer or other electronic device, consisting of a flat, rectangular piece of rigid material, commonly fiberglass, from which all conducting material except the desired circuits is etched, and to which integrated circuits and other electronic components are connected.

**processor:** The hardware component of a computer that performs the actual computation by directly executing instructions represented in machine language and stored in main memory.

**program:** (1) A set of instructions describing actions for a computer to perform in order to accomplish some task, conforming to the rules and conventions of a particular programming language. (2) To write a program.

**programming language:** A set of rules or conventions for writing programs.

**prompt:** To remind or signal the user that some action is expected, typically by displaying a distinctive symbol, a reminder message, or a menu of choices on the display screen.



**prompt character:** A text character displayed on the screen to prompt the user for some action. Often also identifies the program or component of the system that is doing the prompting; for example, the prompt character `]` is used by the Applesoft BASIC interpreter, `>` by Integer BASIC, and `*` by the system Monitor program.

**prompt message:** A message displayed on the screen to prompt the user for some action.

**protocol:** A predefined exchange of control signals between devices enabling them to prepare for and carry out coordinated data transfers.

**push:** To add an entry to the top of a stack.

**radio-frequency modulator:** A device for converting the video signals produced by a computer to a form that can be accepted by a television receiver.

**RAM:** See **random-access memory**.

**random-access memory:** Memory in which the contents of individual locations can be referred to in an arbitrary or random order.

**raster:** The pattern of parallel lines making up the image on a video display screen. The image is produced by controlling the brightness of successive dots on the individual lines of the raster.

**read:** To transfer information into the computer's memory from a source external to the computer (such as a disk drive or modem) or into the computer's processor from a source external to the processor (such as the keyboard or main memory).

**read-only memory:** Memory whose contents can be read but not written; used for storing firmware. Information is written into read-only memory once, during manufacture; it then remains there permanently, even when the computer's power is turned off, and can never be erased or changed. Compare **read-write memory**, **random-access memory**, **write-only memory**.

**read-write memory:** Memory whose contents can be both read and written; often misleadingly called *random-access memory*, or *RAM*. The information contained in read-write memory is erased when the computer's power is turned off, and is

permanently lost unless it has been saved on a more permanent storage medium, such as a disk. Compare **read-only memory**, **random-access memory**, **write-only memory**.

**receive data register:** A read-only register in each serial port ACIA (at location \$C098 for port 1 and \$C0A8 for port 2) that stores the most recent character successfully received.

**register:** A location in a computer processor where an item of information, such as a byte, is held and modified under program control. Registers in the 65C02 microprocessor include the accumulator (A), two index registers (X and Y), the stack pointer (S), the processor status register (P), and the program counter (PC). The PC register holds two bytes (sixteen bits); the other registers hold one byte (eight bits) each.

**remote:** Too distant for direct connection using wires or cables only.

**Request To Send:** An RS-232-C signal from a DTE to a DCE to prepare the DCE for data transmission.

**return address:** The point in a program to which control returns on completion of a subroutine.

**RF modulator:** See **radio-frequency modulator**.

**RI:** See **Ring Indicator**.

**rigid disk:** A disk made of a hard, nonflexible material. Compare **flexible disk**.

**Ring Indicator:** An optional RS-232-C signal from a DCE to a DTE that indicates the arrival of a call.

**ROM:** See **read-only memory**.

**routine:** A part of a program that accomplishes some task subordinate to the overall task of the program.

**RS-232-C:** A standard created by the Electronic Industries Association (EIA) to allow devices of different manufacturers to exchange serial data—particularly via telephone lines.

**RTS:** See **Request To Send**.

**run:** (1) To execute a program. (2) To load a program into main memory from a peripheral storage medium, such as a disk, and execute it.

**save:** To transfer information from main memory to a peripheral storage medium for later use.



**screen:** See **display screen**.

**scroll:** To change the contents of all or part of the display screen by shifting information out at one end (most often the top) to make room for new information appearing at the other end (most often the bottom), producing an effect like that of moving a scroll of paper past a fixed viewing window. See **viewport**, **window**.

**serial interface:** An interface in which information is transmitted sequentially, one bit at a time, over a single wire or channel. Compare **parallel interface**.

**setup time:** The amount of time a signal must be valid in advance of some event; compare **hold time**.

**silicon:** A non-metallic, semiconducting chemical element from which integrated circuits are made.

**soft switch:** A means of changing some feature of the Apple IIc from within a program; specifically, a location in memory that produces some special effect whenever its contents are read or written.

**software:** Those components of a computer system consisting of programs that determine or control the behavior of the computer. Compare **hardware**, **firmware**.

**source code:** See **source program**.

**source program:** The original form of a program given to a language translator such as a compiler or assembler for conversion into another form; sometimes called *source code*. Compare **object program**.

**space character:** A text character whose printed representation is a blank space, typed from the keyboard by pressing the SPACE bar.

**SPACE parity:** A bit of value 0 appended to a binary number for transmission. The receiving device can look for this value on each character as a means of error checking.

**stack:** A list in which entries are added or removed at one end only (the top of the stack), causing them to be removed in LIFO (last-in-first-out) order.

**start bit:** A transition from a MARK signal to a SPACE signal for one bit-time, indicating that the next string of bits represents a character.

**status register:** A register in an ACIA (at location \$C099 for port 1 and \$C0A9 for port 2) that stores the state of two of the RS-232-C signals and the state of the transmit and receive data registers, as well as the outcome of the most recent character transfer.

**stop bit:** A MARK signal following a string of data bits (or their optional parity bit) to indicate the end of a character.

**string:** An item of information consisting of a sequence of text characters.

**strobe:** (1) An event, such as a change in a signal, that triggers some action. (2) A signal whose change is used to trigger some action.

**subroutine:** A part of a program that can be executed on request from any point in the program, and which returns control to the point of the request on completion.

**television receiver:** A display device capable of receiving broadcast video signals (such as commercial television) by means of an antenna. Can be used in combination with a radio-frequency modulator as a display device for the Apple IIc computer. Compare **video monitor**.

**television set:** See **television receiver**.

**terminal:** A device consisting of a typewriterlike keyboard and a display device, used for communicating between a computer system and a human user. Personal computers such as the Apple IIc typically have all or part of a terminal built into them.

**terminal mode:** An operating state of the Apple IIc communication port in which the firmware makes the computer act like a simple ASCII terminal.

**text:** (1) Information presented in the form of characters readable by humans. (2) The display of characters on the Apple IIc's display screen. Compare **graphics**.

**text window:** An area on the Apple IIc's display screen within which text is displayed and scrolled.

**transistor-to-transistor logic:** (1) A family of integrated circuits used in computers and related devices. (2) A standard for interconnecting such circuits that defines the voltages used to represent logical zeros and ones.

**transmit data register:** A write-only register in one of the serial port ACIAs (at location \$C098 for port 1 and \$C0A8 for port 2) that holds the current character to be transmitted.

**troubleshoot:** To locate and correct the cause of a problem or malfunction in a computer system. Typically used to refer to hardware-related problems; compare **debug**.

**TTL:** See **transistor-to-transistor logic**.

**unary operator:** An operator that applies to a single operand; for example, the minus sign (-) in a negative number such as -6 is a unary arithmetic operator.

**user:** The person operating or controlling a computer system.

**user interface:** The rules and conventions by which a computer system communicates with the person operating it.

**vector:** (1) The starting address of a program segment, when used as a common point for transferring control from other programs. (2) A memory location used to hold a vector, or the address of such a location.

**video:** (1) A medium for transmitting information in the form of images to be displayed on the screen of a cathode-ray tube. (2) Information organized or transmitted in video form.

**video monitor:** A display device capable of receiving video signals by direct connection only, and which cannot receive broadcast signals such as commercial television. Can be connected directly to the Apple IIc computer as a display device. Compare **television receiver**.

**viewport:** All or part of the display screen, used by an application program to display a portion of the information (such as a document, picture, or worksheet) that the program is working on. Compare **window**.

**warm start:** The process of restarting the Apple IIc after the power is already on, without reloading the operating system into main memory and often without losing the program or information already in main memory. Compare **cold start**.

**window:** The portion of a collection of information (such as a document, picture, or worksheet) that is visible in a viewport on the display screen; compare **viewport**.

**word:** A group of bits of a fixed size that is treated as a unit; the number of bits in a word is a characteristic of each particular computer.

**wraparound:** The automatic continuation of text from the end of one line to the beginning of the next, as on the display screen or a printer.

**write:** To transfer information from the computer to a destination external to the computer (such as a disk drive, printer, or modem) or from the computer's processor to a destination external to the processor (such as main memory).

**X register:** One of the index registers in the 65C02 microprocessor.

**Y register:** One of the index registers in the 65C02 microprocessor.

**zero page:** The first page (256 bytes) of the Apple IIc's memory, also called *page zero*. Since the high-order byte of any address in this page is zero, only the low-order byte is needed to specify a zero-page address; this makes zero-page locations more efficient to address, in both time and space, than locations in any other page of memory.



# *Bibliography*



*Apple II Monitors Peeled.* Cupertino, Calif.: Apple Computer, Inc., 1978.

Currently not updated for Apple IIe and IIfx, but a good introduction to Apple II series input/output procedures; also useful for historical background.

*Apple IIe Design Guidelines.* Cupertino, Calif.: Apple Computer, Inc., 1982.

*Addendum to the Design Guidelines.* Cupertino, Calif.: Apple Computer, Inc., 1984.

*Apple IIe Reference Manual.* Cupertino, Calif.: Apple Computer, Inc., 1982.

*Applesoft BASIC Programmer's Reference Manual, Volumes 1 and 2.* For the Apple II, IIe, and IIfx. Cupertino, Calif.: Apple Computer, Inc., 1982.

The version that applies to both the Apple IIe and the Apple IIfx has Apple product number A2L0084 (Vol. 1) and A2L0085 (Vol.2).

*Applesoft Tutorial.* Cupertino, Calif.: Apple Computer, Inc., 1982.

Leventhal, Lance. *6502 Assembly Language Programming.* Berkeley, Calif.: Osborne/McGraw-Hill, 1979.

*Synertek Hardware Manual*. Santa Clara, Calif.: Synertek Incorporated, 1976.

Does not contain instructions new to 65C02, but is the only currently available manufacturer's hardware manual for 6500 series microcomputers.

*Synertek Programming Manual*. Santa Clara, Calif.: Synertek, Incorporated, 1976.

The only currently available manufacturer's programming manual for 6500 series microcomputers.

Watson, Allen, III. "A Simplified Theory of Video Graphics, Part I." *Byte* Vol. 5, No. 11 (November, 1980).

———. "A Simplified Theory of Video Graphics, Part II." *Byte* Vol. 5, No. 12 (December, 1980).

———. "More Colors for Your Apple." *Byte* Vol. 4, No. 6 (June, 1979).

———. "True Sixteen-Color Hi-Res." *Apple Orchard* Vol. 5, No. 1 (January, 1984).

Wozniak, Steve. "System Description: The Apple II." *Byte* Vol. 2, No. 5 (May, 1977).



# *Index*

References to entries in Volume 2 are in square brackets [ ].

### ***Cast of Characters***

\* (asterisk) 179  
 \ (backslash) 59  
 \_ (blinking underscore cursor) 154  
 > (greater than sign) 59  
 ? (question mark) 58, 59  
 ] (right bracket) 59

### **A**

A register 17  
 accumulator 17  
 ACIA 134, 148, 253-262, [63]  
     block diagram 255  
     interrupts [60]  
 address bus 12, 213  
 AKD 218-219  
 ALTCHAR 104-105, 218, [73]  
 alternate character set 68, [73]  
 ALTZP 25, 26, 216, [46]  
 analog inputs 176, [68]  
 annunciator outputs [76]  
 ANSI [84]  
 any-key-down 79, 229  
     flag [66]  
 Apple Extended 80-Column  
     Text Card [67, 74]  
 Apple Language Card [64]

Apple II series differences [60-78]  
 Apple IIc  
     block diagram 210  
     care of 205-206  
     differences from Apple IIe [61-78]  
     expansion 2  
 Apple IIe ROMs [72]  
 Applesoft & commands 52  
 Applesoft BASIC 59, [16-18, 40]  
     BASIC interpreter 24  
 Applesoft interpreter 21, 224-225  
 arithmetic, hexadecimal 193  
 ASCII [71, 83, 86-87]  
     character set 79, [97, 114-122]  
 assemblers 199  
 assembly language, and mouse 171  
 asterisk (\*) 179  
 automatic line feed 131, 145  
 automatic repeat 3  
 Autostart ROM [69]  
 auxiliary memory screen holes  
     135-136, 149-150  
     *See also* screen holes  
 auxiliary RAM 20  
 AUXMOVE *See* MOVEAUX  
 AY-3600-type keyboard decoder 229

## B

- B command 131, 144
- back panel 8, 9
- backslash (\) 59, 62
- backspace 62
- bank 25
- bank-switched memory 22, [64, 69]
- BANK2 216
- BASIC 130, 163, 175-177, 179, 180, 192, [114]
  - and assembly language support 171
  - and hand controls 173
  - and mouse 163, 172
- BASIC*S disk [39, 69]
- baud rate 137, 258
- BCLK 256
- BELL 84
- BELL1 84
- BIT instruction [3]
- bits [103]
- blanking intervals 233
- blinking underscore cursor ( \_ ) 154
- block diagrams
  - ACIA 255
  - Apple IIc 210
- BREAK 132, 137, 145
- break instructions [48]
- BREAK signal [75]
- BRK 75, 189, [43]
- buffer 59
  - serial I/O [75]
- built-in diagnostics [62]
- built-in disk drive 8
- built-in self-tests [65]
- button interrupt mode 164, 167
- bypassing firmware [58-60]
- byte(s) [103, 104]
  - power-up 51

## C

- C06X 267
- C07X 217
- C3COUT1 55, 64

- C3KEYIN 55
- CALL statement 179
- Canadian keyboard [91]
- cancel line 62
- CAPS-LOCK 4, 79, [84]
- card(s) [74, 75]
- care of computer 205-206
- carriage return 139, 152
- carrier 137
- CAS (column-address strobe) 228
- cassette input and output [67-68, 77]
- certifications [99]
- CH (cursor horizontal) 63
- changing memory contents 184
- changing registers 190
- character(s)
  - flashing 68
  - generator 241
  - inverse 68
  - normal 68
  - sets [71, 73]
- chips, custom [78]
- clamping boundaries 171
- CLAMP MOUSE 168
- CLEAR MOUSE 168
- CLEOLZ 116
- clock 211
  - master 213
  - system 213
- CLREOL 116
- CLREOP 116
- CLRSCR 117
- CLRTOP 117
- code conversions [114-122]
- cold-start procedure 49, 50
- colors
  - high-resolution 243
  - low-resolution 242, [63]
- command character 146, [75]
- command register 134, 148, 260
- Communication Card [74]
- communication port 141
- comparing data in memory 188-189



- connector(s)
  - back panel 8-9
  - game [78]
  - power 207
  - serial port 257
- CONTINUE BASIC command 192
- CONTROL 4, 79, 229
  - transferring 42-43
- control characters 64
- control register 134, 148, 258-259
- CONTROL-A, as command character 143
- CONTROL-C [53]
- CONTROL-H 62
- CONTROL-I, as command character 130, 132
- CONTROL-K, as command character 193
- CONTROL-P 56, 126, 142
  - as command character 193
- CONTROL-R 155
- CONTROL-S [53]
- CONTROL-T 156, 159
- CONTROL-X 62
- CONTROL-Y 197
  - commands 52
- CONTROL RESET 50
- conversion, number [106]
- COUT 55, 117, 191
- COUT1 55, 68, 117
- CP/M [40]
- CPU *See* 65C02
- CR *See* carriage return
- CREF 220, 221, 251
- CROUT 117
- CROUT1 117
- CSW 56, 70, 104
- cursor 58, 130, 143, 193
  - blinking underscore ( \_ ) 154
  - flashing checkerboard 55
  - flashing question mark 130, 143
  - inverse solid 55

- custom chips [78]
- custom integrated circuits 215-223
- CV 63

## D

- D command 131, 144
- data, transferring 41-42
- data bits 137
- data bus 213
- Data Carrier Detect [60]
- data format 137, 138, 144, 151
- data inputs 21
- Data Set Ready *See* DSR
- Data Terminal Ready *See* DTR
- DCB 261
- DCD [60]
- decimal, negative [107]
- device signature 72
- DEVNO [21]
- DHIRES 49, 104, 106, 107, 166
- diagnostics, built-in [62, 65]
- differences among Apple II's [61-78]
- disable MouseText 65
- DISK 221, 222
- disk
  - controller cards [74]
  - controller unit *See* IWM
  - input and output 124-126
  - I/O firmware entry points 20
- disk drive 8
  - connector 252
  - port [50]
  - speed 13
- disk-use light 6, [71]
- display
  - address mapping 235-238
  - inverse 191
  - memory addressing 234
  - memory switches 43-47
  - modes 104-108, 239-247
  - normal 191
  - page maps 108-114
  - pages 102-103

- DISVBL 166
- DISXY 166
- DMA transfers [70]
- DOS 126, 130, 143, 179, 180, [39, 69]
  - interrupts [42]
  - zero page use [16-18]
- double-high-resolution 245
  - graphics [74]
  - colors 100-101
- drive, external, startup 126
- drive motor 49
- DSR 256, 261, [60]
- DSR1B 257
- DSR2B 257
- DTR 260
- DTR1B 257
- DTR2B 257
- dumb terminal 159
- Dvorak keyboard 6, [88]
- dynamic-RAM refreshment and timing 226-229

## E

- echo 131, 145, 155, 260
- EIA standard 258
- 80 columns 65, 93
- 80/40 column switch 5
- 80COL 104, 105, 107, 108, 218, 219, 220
- 80STORE 39, 44, 45, 104, 105, 107, 108, 216, 238, 241
- electrical power 206
- EN80 217
- enable MouseText 65
- ENBVBL 166
- ENBXY 166
- ENCLCRAM 216
- English keyboard [90]
- enhanced video firmware 20, 224
- enter terminal mode 145
- entry points, firmware [31-36]
- environmental specifications 205-206

- ESC** 4
- ESC** **4** 61
- ESC** **B** 61
- escape codes 60
- escape sequences 4
- even-parity [114]
- EXAMINE command 190
- examining memory contents 181
- examining registers 190
- expansion ROM space 73
- Extended 80-Column Text Card [64]
- external drive startup 126
- external interrupts [55]
- external power connector 207
- EXTINT 256, [55, 60]

## F

- FCC [99]
- firmware 12
  - entry points [30-36]
  - listings [126-215]
  - locations [30-36]
  - protocol 71, 134, 148
  - video routines 115-123
- flag inputs 21
- FLASH 256
- flashing characters 68
- flashing checkerboard cursor 55
- flashing power light 6
- forced cold start 50
- 14M 215, 220, 221
- FORTTRAN [41]
- 40 columns, switching to 80 5
- 40-column 65, 93
- 48K memory 34, 35, 39
- framing errors 258
- French keyboard [91-92]
- full duplex 156-158

## G

- GAME I/O connector [76]
- game input 267
- game paddles *See* hand controls
- GAMESW0 268
- GAMESW1 268
- General Logic Unit (GLU) 13
- German keyboard [93]
- GETLN 58-62, 180
- GETLN1 59, 82
- GETLNZ 82
- GLU 221
- GND 257
- GO command 189, 190, 192, 198
- graphic bits [109]
- graphics mode 96-102
- greater than sign (>) 59

## H

- half duplex 155
- hand control 8, 173-178
  - circuits 269
  - connector 174
  - input [76]
  - signals 270
- hand controller 267
- handle 9, 206
- hardware
  - accesses 21
  - addresses [66]
  - locations 181, [15]
  - page locations 164
- headphones 232
- heat 206
- hexadecimal [106]
  - arithmetic 193

- high-resolution 97
  - colors 243
  - display 243
  - double 245
  - graphics colors 98-99
  - Page 1 37
  - Page 2 38
- HIRES 44, 45, 104, 105, 107, 216, 218, [67]
- HLINE 117
- HOME 118
- HOMEMOUSE 168
- HRP1 37
- HRP1X 37, 45
- HRP2 45
- HRP2X 38
- humidity 205

## I

- I command 131, 145, 158
- I/O firmware, video routines 120-123
- I/O links 55
- icons 68
- identification bytes 71
- IEC [99]
- IN#2 143, 154
- IN#n 56, 70
- index registers 17
- INH 217
- INITMOUSE 169
- input and output, disk 124-126
- input buffer (page \$02) 36
- Input/Output Unit (IOU) 13, 215, 218-219, [78]
- instruction cycle times [63]
- Integer BASIC 59, [16-18, 41, 69]
- Integrated Woz Machine (IWM) 13
- internal converter 208
- internal voltage converter 206

- interrupt(s) 24, 75, 260, [40-60, 70]
  - ACIA [49]
  - Apple II and [42]
  - Apple II Plus and [42]
  - Apple IIe and [43]
  - disk drive port [49]
  - DOS and [42]
  - keyboard [52-53]
  - Monitor and [42]
  - mouse [49]
  - Pascal and [42]
  - 65C02 and [43]
  - 6551 [49]
  - vertical blanking [49]
- interrupt handler(s)
  - mouse 163
  - user's [57]
- interrupt requests 52
- interrupt vector [43-44]
- interrupt-handling sequence [45]
- inverse 65
  - characters 68
  - display 191
  - solid cursor 55
- INVERSE command 191
- invoking the monitor 179
- IOREST [36]
- IORTS [36]
- IOSAVE [36]
- IOU (Input/Output Unit) 13, 215, 218-219, [78]
- IOUDIS 49, 104, 106, 166, [67, 68]
- IOUSELIO 219
- IRQ 75, 156, 219, [43]
  - handling routine [34]
  - vector [36]
- ISO [84]
  - layout [89]
- Italian keyboard [94]
- IWM (Integrated Woz Machine) 13, 222

## J

- jack 7
- JMP \$C600 126
- JMP indirect instruction [3]
- joysticks *See* hand controls

## K

- K (1024) 17
- K command 131, 145
- KBD 217
- keyboard 229-231
  - buffer [52-53]
  - character decoder 225
  - circuit diagram 230
  - data [66]
  - input buffer 37
  - interrupts [52, 53]
  - layout [71]
    - ANSI [90]
    - British *See* English
    - Canadian [91-92]
    - Dvorak [88]
    - English [90]
    - French [91-92]
    - German [93]
    - ISO [90]
    - Italian [94-95]
    - Sholes [85]
    - Western Spanish [96]
  - signals 231
  - strobe 79, 229, [50, 66]
  - switch 5
  - standard 5
- KEYIN 55, 57, 58
- KSTRB 77, 219, 256
- KSW 56, 57, 70, 104

## L

- L command 131, 145
- LANGSW 256
- LDPS 220, 241, 251

- line feed 145, 152
  - automatic 131
- line length 136, 150
- line voltage 205
- line width 139, 144
- LIST command 199
- local 154
- low-resolution
  - colors 242
  - display 242
  - graphics 96

## M

- machine identification [63]
- main memory screen holes 135-136, 149, 150
- main RAM 20
- MARK (1) 132
- MARK parity 138, [114]
- master clock 213
- maximum current drain 252
- memory
  - addressing 223-229
  - bank-switched 22
  - bus organization 224
  - comparing data in 188-189
  - display switches 43-47
  - dump 182-184
  - examining contents 181
  - 48K 34
  - map 18, [15-28]
  - moving data in 186-188
  - organization [64]
  - state [48]
  - switches, display 43-47
- Memory Management Unit *See* MMU
- microprocessor, 65C02 12, 15
- mini-phone jack 7
- MIXED 105, 107, 218, [67]
- mixed-modes displays 102
- MMU 13, 215-217, 267, 271, [78]
- mnemonic 199
- modem 8, 151
- modes, display 239-247
- monitor 8, 24, 59, 179-203, 224
  - entry point [36]
  - interrupts and [42]
  - output 248
  - register commands 189-190
  - ROM [69]
  - video routines 115
  - zero page use [15]
- mouse 8, 160-174, [49-50]
  - BASIC and 163, 172
  - Pascal and 171
  - button 171
    - interrupt mode 164
    - signals 266
  - clamping boundaries 171
  - connector 264
  - direction [59]
  - firmware 167
  - firmware entry points 20
  - hardware locations 164-167
  - input 262, [76]
  - interrupt handler 165
  - interrupts [58]
  - movement interrupt mode 163
  - operating modes 163
  - port 161-174
  - transparent mode 163
  - waveform 263
  - X direction 167
  - Y direction 167
- MOUSEID 264
- MouseText 65, 68-69, 90-91, [73, 114]
- MOUX1 167
- MOUY1 167
- MOVE command 186-188, 195, [36]
- MOVEAUX 41-42
- movement/button interrupt mode 164, 167
- movement interrupt mode 163, 167
- moving data in memory 186-188
- MSLOT [21]
- MSW 264



## N

N command 131, 145, 156  
n CONTROL-K 56  
NE556 265, 271, [77]  
negative decimal [107]  
NEWIRQ [34]  
nibble [104]  
NMI vector [36, 43]  
non-maskable interrupts 52  
NORMAL command 191  
normal characters 65, 68  
normal display 191  
NTSC 87, 233, 242, 248, 251  
#6 130  
#7 143  
#8 143

## O

odd-parity [114]  
old monitor ROM [62]  
1 CONTROL-P 130  
1VSOUND 251  
⓪ 4, 82  
operand 199  
operating systems [39-40]  
operating temperature 205  
output and input, disk 124-126  
output jack 232

## P

P command 132, 145  
P register 17  
paddle(s) 267  
    button 0 268  
    button 1 268  
    inputs [68, 76]  
    timing circuit [77]  
page 18  
page \$02 (input buffer) 36  
page \$03 36  
page \$04 36  
page \$08 37

page 0 18  
page zero 24  
page 1 18  
PAGE2 44-45, 105, 107-108, 216,  
    238, 241, [46-48, 67]  
page three [19]  
page 8, auxiliary RAM [52]  
PAL 233  
parity 145  
    bit(s) 138, 262  
    checking 260  
Pascal 67, 126, 130, 134, 170, [114]  
    ID byte 134, 148  
    interrupts and [42]  
    language [41]  
    operating system [40]  
PC (program counter) 16  
PCAS 220  
PDL0 176  
PDL0/XMOVE 219  
PDL1 176  
PEEK [40]  
peripheral identification numbers [112]  
peripheral-card memory space [65-66]  
peripheral-card ROM space [65]  
phone jack 7  
PIN numbers [112]  
PINIT 72, 121, 134, 148  
PLOT 118  
plotter 8  
POKE [40]  
ports 70, [70]  
POSMOUSE 168  
power 8  
    connector 207  
    consumption 207, 208  
    light 6, [71]  
    requirements 206  
    supply [100]  
power-on light [71]  
power-up byte 51



- PR#1 130
- PR#2 143, 154
- PR#6 126
- PR#n 56, 70
- PRAS 217, 219, 220, 251
- PRBL2 118
- PRBYTE 118
- PREAD 72, 121, 134, 148, 177
- PRERR 118
- PRHEX 118
- primary character set 68, [73]
- printer 8
- PRINTER: 130
- processor status register 17
- ProDOS 126, 130, 143, 180, [39, 63]
- program counter (PC) 16, 201
- prompt 58, 154
  - characters 59
- PRTAX 118
- PSTATUS 72, 123, 134, 148
- PTRIG 166
- published entry points [32-36]
- pull* from stack 17
- push* onto stack 17
- PWRITE 72, 121, 134, 148

## Q

- Q command 145
- Q3 215, 217, 219
- question mark (?) 58, 59
- quit terminal mode 145

## R


- R command 132
- R/W 217, 219, 221, 257
- RAD-RA7 217
- RAM 17
  - addressing 226-229
  - locations [15]
- RAMRD 38, 39, 43, 44, 216, [46]
- RAMWRT 38, 39, 43, 44, 216, [46]

- random number 58
- random-access memory (RAM) 17
- RAS (row-address strobe) 228
- RD1B 257
- RD63 167
- RD80COL 105
- RD80STORE 105
- RDALTCHAR 105
- RDALTZP 26
- RDBNK2 26
- RDCHAR 82
- RDCRAM [46]
- RDDHIRES 106
- RDHIRES 45, 105
- RDIODIS 106, 166
- RDKEY 55, 57
- RDLCRAM 26
- RDMIXED 105
- RDPAGE2 105
- RDRAMRD 39
- RDRAMWRT 39
- RDTEXT 105
- RDTNO 167
- RDVBLMSK 166
- RDXYMSK 166
- RDYQEDGE 166
- read-only memory (ROM) 17
- READMOUSE 163, 168, [51-52]
- receive register 262
- registers 15, 213
  - examining 190
- relative humidity 205
- REMIN 143
- remote 154, 159
- remote device 145
- REMOUT 143
- (REPT) key [71]
- Request to Send See RTS
- (RESET) key 4, 79, 82, 221, 113, 256, [71]
- reset port 1 132
- reset port 2 145
- reset routine 48
- reset vector 49-51, [36]
- (RETURN) [84]


- retype 62
- RF modulator 233
- RGB monitor 245
- rollover 3
- ROM 17
- ROM addressing 224-225
- ROMEN2 217
- RS-232 129
- RSTVBL 166
- RSTXINT 166, 216
- RSTXY 166
- RSTYINT 166, 216
- RTS instruction 260, [36]

## S

- S command 132
- S register 17
- safety instructions 207, [99]
- schematic diagrams 271-276
- scratch-pad RAM [65]
- screen holes 36, 73, 74, 133, 134, 136, 149, 171-173, [20-22, 47]
- SCRN 119
- scroll 65
- SEGA 218
- SEGB 218, 220, 251
- self-tests *See* diagnostics, built-in
- SER 221, 256
- serial buffering [55]
- serial data transfer [57]
- serial firmware [50]
- serial I/O buffers [75]
- serial I/O port 128-159
- serial input buffer 37
- Serial Interface Card [74]
- serial interrupts [55, 56]
- serial port circuits 254
- serial port 1 20, 129-139
- serial port 2 20, 141-159
  - command character 143, 146
  - command character hardware locations 130, 132, 134
  - firmware protocol 147
  - hardware locations 148
  - initial characteristics 130, 147
- SEROUT 251
- SERVEMOUSE 163, 168, [51]
- SETCOL 119
- SETMOUSE 167-168, [50-51]
- SETPWRC 51
- 7M 220, 223
- SHIFT** key 79, 229, [84]
- shift-key mod [68]
- Sholes keyboard 5
- signature byte 134, 148, 170
- simplified keyboard (Dvorak) [88]
- 6** 126
- 65C02 12, 15, [63]
  - address bus 213
  - addressing modes [10]
  - block diagram 211
  - clock 211
  - cycle time [1, 2]
  - data bus 213
  - data sheet [5-13]
  - differences from 6502 211, [1-3, 6-7]
  - execution time [1-2]
  - instruction set [12-13]
  - opcodes [12]
  - registers 213
  - signal descriptions [11]
  - timing diagram [8]
  - timing signals 214-215
- 6502 versus 65C02 211
- 6551 Asynchronous Communication Interface Adapters *See* ACIA
- slot 7 drive 1 [74]
- SLOT3ROM [66]
- SLOTXROM [66]

- slots 70
  - versus ports [70]
- soft switches 22, 215, 218, 221
-  82
- SPACE (0) 132
- SPACE parity 138, [114]
- speaker 83-84, [67]
  - external 7
  - output jack 232
  - volume control 232
- SPKR 219
- stack 24, [42, 46]
- stack pointer 17
- standard I/O links 55
- standard keyboard 5
- start bit 137
- status register 134, 148, 261
- stop bits 137
- stop-list 65
- STORE command 194
- strobe 79
  - inputs 21
- SUD *See* System Utilities Disk
- Super Serial Cards [74]
- SW0 175
- SW1 175
- switch inputs 175, [76]
- switches, soft 22, 215
- SYNC 219, 233, 251
- system clock 213
- system monitor 179-203
- System Utilities Disk* 129, 131, 136, 141, 145, 150, [75, 112]

## T

- T command 145, 154-156, 159
-  [84]
- TD1B 257
- telephone jack 7
- temperature 205, 208

- terminal mode 145, [53]
- TEXT 105, 107, 218, 220, 221, 251, [67]
- text
  - and low-resolution graphics Page 1 36
  - and low-resolution Page 1X 36
  - and screen low-resolution Page 2 37
  - displays 241
  - modes 90-95
  - window 63, 66
- TLP1 36
- TLP1X 36, 45
- TLP2X 37
- toggle switches 22
- transferring control 42-43
- transferring data 41-42
- transmit/receive data register 134, 148
- transmit register 262
- transparent mode 163, 167, 171
- triggering paddle timers [68]

## U

- USA standard keyboard 5
- USER command 197
- user's interrupt handler [57]
- utility strobe [67]

## V

- validity check 49
- VBL [67, 73, 76]
- VBLINT 163, 164, 218, [67, 73]
- VDE [99]
- vectors 55
- ventilation 206
- VERIFY command 188, 196, [36]
- vertical blanking 163, [49, 50, 73]
  - interrupts [68]

VID 248  
VID7M 215, 220  
video  
    counters 233-234  
    display 225  
    display circuits 240  
    display modes 239-247  
    expansion 8  
    expansion connector 249-252  
    expansion output 249  
    output signals 248  
    routines  
        firmware 115-123  
        I/O firmware 120-123  
        monitor 115-119  
VLINE 119  
voltage 205  
    converter 10  
volume control 7, 232

## W

WAIT [36]  
warm-start procedure 50  
Western Spanish keyboard [96]  
WNDW 219, 233, 251  
word [106]  
Woz Integrated Machine 13, 222

## X

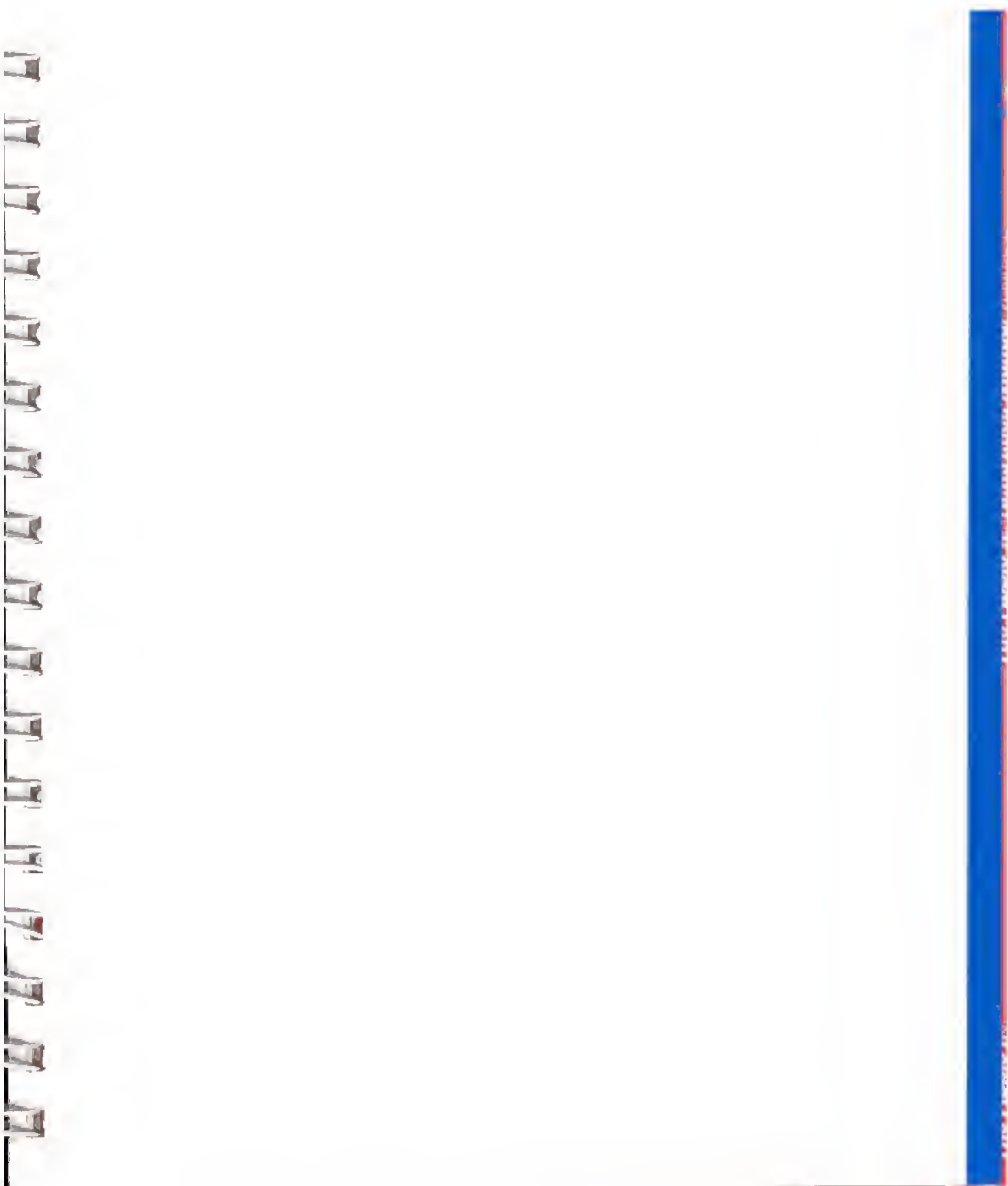
X register 17  
X0 215, 218, 262, 264  
X1 215, 263, 264  
XFER 41, 42  
XINT 164, [66, 67]  
XOEDGE 166

## Y

Y register 17  
Y0 218, 262, 264  
Y1 263, 264  
YINT 164, [66, 67]  
YMOVE 219  
YOEDGE 166

## Z

Z command 132, 139  
zap 132, 139, 145  
zero page 24, 184











Apple Computer, Inc.  
20525 Mariani Avenue  
Cupertino, California 95014  
(408) 996-1010  
TLX 171-576

030-1022-A  
1984 Apple Computer, Inc.  
Printed in U.S.A.